

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-43176

(P 2 0 0 1 - 4 3 1 7 6 A)

(43) 公開日 平成13年2月16日 (2001.2.16)

(51) Int. Cl. <sup>7</sup>

G06F 13/00

識別記号

620

F I

G06F 13/00

620

ターマコード (参考)

審査請求 未請求 請求項の数10 O L (全38頁)

(21) 出願番号 特願2000-176806 (P 2000-176806)

(22) 出願日 平成12年6月13日 (2000.6.13)

(31) 優先権主張番号 09/333421

(32) 優先日 平成11年6月15日 (1999.6.15)

(33) 優先権主張国 米国 (U S)

(71) 出願人 591064003

サン・マイクロシステムズ・インコーポレーテッド

SUN MICROSYSTEMS, INCORPORATED

アメリカ合衆国 94303 カリフォルニア州・パロアルト・サンアントニオロード・901

901 SAN ANTONIO ROAD  
PALO ALTO, CA 94303, U. S. A.

(74) 代理人 100064621

弁理士 山川 政樹

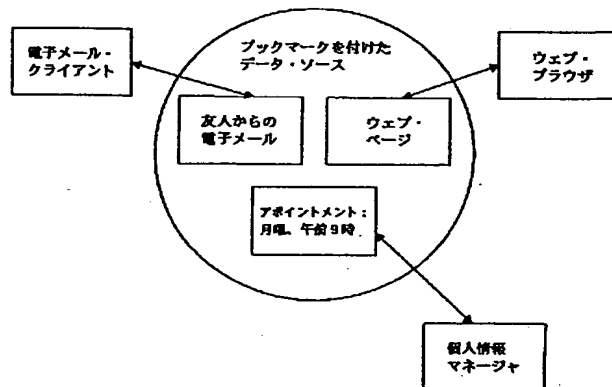
最終頁に続く

(54) 【発明の名称】 小フットプリント・デバイス・アプリケーション用の高性能ブックマーク

(57) 【要約】

【課題】 高性能携帯電話、携帯情報端末などの小フットプリント・デバイスのユーザが、様々なタイプのオブジェクトおよび/またはデータ・ソースを参照するブックマークを作成し、格納することができるシステムを提供する。

【解決手段】 各ブックマークは、URLを使用してオブジェクト/データ・ソースを参照する。このブックマーク・システムはオープン・エンドにすることができ、実質的にいかなるタイプのオブジェクトまたはデータ・ソースにもブックマークを付けることができる。後にユーザまたはアプリケーションがブックマークを使用してそれぞれのデータ・ソースを参照し、表示や編集など、あるタイプのアクションをそのデータ・ソースに対して実行することができる。



## 【特許請求の範囲】

【請求項1】 データ・ソースへの持続的参照を作成するシステムであって、

処理ユニットおよびシステム・メモリを含む小フットプリント・デバイスと、

前記システム・メモリに格納され、コンピューティング・サービスを実装するプログラム・モジュールをサポートするソフトウェア・フレームワークと、

1つまたは複数の前記プログラム・モジュールに実装され、ユーザが前記データ・ソースを選択することに応答してデータ・ソースへの持続的参照を作成するように動作可能である第1のコンピューティング・サービスと、1つまたは複数の前記プログラム・モジュールに実装され、前記持続的参照を使用して前記データ・ソースにアクセスするように動作可能である第2のコンピューティング・サービスとを含むシステム。

【請求項2】 前記データ・ソースをカプセル化するエンティティを作成するように動作可能な起動フレームワークをさらに含み、第1のコンピューティング・サービスによって作成された前記持続的参照は、前記データ・ソースをカプセル化する前記エンティティを参照する請求項1に記載のシステム。

【請求項3】 前記起動フレームワークがさらに、前記第2のコンピューティング・サービスを呼び出して、前記データ・ソースに対して操作を実行するように動作可能である請求項2に記載のシステム。

【請求項4】 前記起動フレームワークがJava Beans 起動フレームワークである請求項3に記載のシステム。

【請求項5】 前記ソフトウェア・フレームワークが、300キロバイト未満のメモリを必要とする請求項1に記載のシステム。

【請求項6】 前記小フットプリント・デバイスが、携帯情報端末(PDA)、携帯電話、および全世界測位システム(GPS)受信機からなるグループからの小フットプリント・デバイスである請求項1に記載のシステム。

【請求項7】 前記小フットプリント・デバイスが、ゲーム・コンソール、ウェアラブル・コンピューティング・デバイス、セット・トップ・ボックス、および電子ブック・デバイスからなるグループからの小フットプリント・デバイスである請求項1に記載のシステム。

【請求項8】 前記小フットプリント・デバイスが、2メガバイト未満のメモリを含む請求項1に記載のシステム。

【請求項9】 前記小フットプリント・デバイスが、20平方インチより小さいディスプレイ画面を含む請求項1に記載のシステム。

【請求項10】 前記小フットプリント・デバイスが、専らバッテリー駆動型である請求項1に記載のシステム。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】 本発明は、一般にコンピュータ・アプリケーション・プログラムおよび小さなフットプリントのデバイスに関する。より詳細には、本発明は、その小フットプリント・デバイス上で動作するアプリケーションおよびサービスによって使用するための様々なタイプのデータ・ソース・オブジェクトにブックマークを付けるシステムおよび方法を含む。

## 【0002】

【従来の技術】 「高性能」の小フットプリント・デバイスの分野は、急速に成長し、変化している。小フットプリント・デバイスとしては、ハンドヘルド・コンピュータ、携帯情報端末(PDA)、携帯電話、全世界測位システム(GPS)受信機、ゲーム・コンソール、およびさらに多数のこのようなデバイスなどがある。これらのデバイスは、よりインテリジェントにかつ相互接続されるようになりつつある。Sun Microsystems社のJini<sup>TM</sup>などの技術およびOpen Service Gateway Initiative(OSGI)などのイニシアチブが、従来のコンピュータ・ネットワークの概念を、小フットプリント・デバイスを含めるように拡張している。

【0003】 このデバイス相互接続の増加によって、新しいタイプのコンピューティング・サービスと、デバイス間ベースおよびデバイス内ベースのサービスのコンピューティング・サービスを統合する新しい方法に対する必要性が出てきた。「サービス」とは、デバイス内あるいはデバイスからアクセス可能に実装されたエンティティであり、人、アプリケーションまたは別のサービスによって使用することができるものである。サービスの概念は幅広く、多数の異なる尺度で考慮することができる。たとえば、サービスとしては、共有印刷、電子メール、テレフォニーなど、周知のネットワーク・ベースのサービスがある。サービスには、ローカル・ネットワーク内のデバイスの電力消費量を制御することができるエネルギー管理サービス、エラーが生じたときにデバイスが情報をサービス技術者に送信することができる診断サービス、保健専門家に急患を即座に通知する健康監視サービスなど、それほど周知でない例もある。

【0004】 サービスには、ローカルのマシンまたはデバイス内に位置する、実行可能なモジュールまたはアプリケーションなどもある。たとえば、ローカル・アプリケーション・プログラムは、サービスを利用して、HTTPサーバ、HTMLレンダ・エンジン・サービス、ブックマーク・サービス、ユーザ・インタフェース・サービスなどと通信することができる。この例では、アプリケーション・プログラムがこれらのサービスを共に使用して、ウェブ・ブラウザ・プログラムを実装することができる。

【0005】今日、多数のサービスおよびアプリケーションを共に単一のフットプリント・デバイスで実行することが、より一般的になりつつある。しかし、メモリ、処理能力および他のリソースが、一般にフットプリント・デバイスでは非常に制限されるため、専用の軽量サービス/アプリケーション・コンテインメント・フレームワークが、所望のサービスおよびアプリケーションの統合を達成するために必要である。コンテインメント・フレームワークは、いかなる種類のフットプリント・デバイス用のいかなるタイプのサービスおよびアプリケーション用のフレームワークをも提供するために、十分柔軟性があり、拡張可能であることも望ましい。さらなる目標は、コンテインメント・フレームワークが、Jini<sup>TM</sup>ネットワークにおけるデバイスに使用可能なサービスなど、オフ・デバイス・サービスと互換性を持ち、統合されるようにすることであることがある。本明細書に記載するコンテインメント・フレームワークは、上述の目標を達成する。

【0006】軽量のコンテインメント・フレームワークは、携帯情報端末、高性能携帯電話などのフットプリント・デバイスで、従来はデスクトップ・コンピューティング環境に関連したタイプの多目的アプリケーション・プログラムを実行することができる。たとえば、Sun Microsystems社から入手可能なPersonal Applicationsスーツは、コンテインメント・フレームワークの一実施形態を中心に構築されている。Personal Applicationsスーツは、Personal Applications Browser、Personal Applications Email ClientおよびPersonal Organizerなど、コンパクトなメモリ効率のよいアプリケーションの統合セットを含んでいる。

【0007】これらのタイプのアプリケーションがフットプリント・デバイスに使用可能になるので、様々なアプリケーションまたはサービスによって使用あるいは実装されるエンティティ、オブジェクトおよびデータ・ソースを統合および抽象化する汎用機構を提供することが、ますます望ましくなる。このタイプの抽象化および統合を可能にするための一手法は、ブックマーク・サービスを使用可能にして、アプリケーション・プログラムまたはサービスで作成あるいは使用することができる、様々なタイプのエンティティ、オブジェクトおよびデータ・ソースのいずれにも対する持続的参照を提供することである。たとえば、ユーザは、特定の電子メール、またはコンタクト・リストにある特定のコンタクト・エントリにブックマークを付けて、後に都合よく参照できるようにすることができる。

【0008】

【発明が解決しようとする課題】このタイプのブックマ

ーク・システムを使用可能にすることは、特にフットプリント・デバイス環境内では望ましい。このようなシステムでは、特定の動作を実行する際の困難さのレベルを著しく低減することができる。これは、ユーザがフットプリント・デバイスで入力する手段がしばしば非常に限定されるからである。しかし、このタイプのブックマーク・システムの実装は、フットプリント・デバイス環境にとって特定の難題でもある。これは、所望の抽象化および一般性を達成するためにあるレベルのソフトウェア・インフラストラクチャが必要だからであり、しかもこのソフトウェア・インフラストラクチャがフットプリント・デバイス上で動作するために十分に軽量でなければならないからである。このタイプのブックマーク・システムをフットプリント・デバイスに実装するシステムおよび方法を、本明細書に記載する。

【0009】

【課題を解決するための手段】本発明は、フットプリント・デバイスのユーザが、様々なタイプのオブジェクトおよび/またはデータ・ソース（本明細書では総称してデータ・ソースと呼ぶ）を参照するブックマークを作成し、格納することができるようにする。各ブックマークは、ユニフォーム・リソース・ロケータ（URL）を含むことができ、これを使用してオブジェクト/データ・ソースを参照することができる。このブックマーク・システムはオープン・エンドにすることができ、実質的にいかなるタイプのオブジェクトまたはデータ・ソースにもブックマークを付けることができる。後にユーザまたはアプリケーションがブックマークを使用してそれぞれのデータ・ソースを参照し、表示や編集など、あるタイプのアクションをそのデータ・ソースに対して実行することができる。

【0010】たとえば、ユーザは、電子メール・クライアント内から特定の電子メールにブックマークを付けることができる。この場合、ブックマークはあるタイプのカプセル化オブジェクトを参照することができる。これはデータ・ソースと呼ばれ、電子メールを表現するデータを含んでいる。したがって、ブックマーク・サービスは、様々なタイプのエンティティをカプセル化することができる起動フレームワークと協調することができる。起動フレームワークは、1つのタイプを1つのデータソースに割り当てることができ、これによってデータ・ソースを他の種類のデータ・ソースから区別する。起動フレームワークは、特定の「動詞」をデータ・ソース・タイプに関連付けることもでき、特定の実行可能モジュールまたはルーチンをデータ・ソース・タイプおよび動詞に関連付けることもでき、そこでこの実行可能モジュールを呼び出して、この動詞によって特定のデータ・ソースに指示されたアクションを実行することができる。

【0011】ブックマーク・サービス・モジュールは、フットプリント・デバイスのアプリケーション/サー

ビス・コンテインメント・フレームワーク内で実行し、他のアプリケーション／サービスによって、特定のデータ・ソースを参照するブックマークを作成して格納するために呼び出すことができる。小フットプリント・デバイス用の軽量のアプリケーション／サービス・フレームワークを、以下に説明する。このブックマーク・サービスは、上述のように、起動フレームワークと共に動作することができる。一実施形態では、ブックマーク・サービスがJava Beans<sup>TM</sup>起動フレームワーク(JAF)と共に動作して、上述の機能性を実装する。

【0012】本発明の他の目的および利点は、以下の詳細な説明を読み、添付の図面を参照することによって明らかになるであろう。

#### 【0013】

【発明の実施の形態】本発明は、様々な修正および代替形式が可能であり、その特定の実施形態を例として図面に示し、本明細書で詳細に説明する。しかし、図面および詳細な説明は、本発明を、開示した特定の形式に限定することを意図するものでなく、逆に、添付の特許請求の範囲によって定義された本発明の精神および範囲内に入るすべての修正、均等物および代替物を包含することを意図するものである。

#### 【0014】図1—データ・ソースの例

図1は、ブックマークを付けることができるデータ・ソースのいくつかの例である。上述のように、ブックマーク・サービスはオープンな起動フレームワークと共に動作することができ、様々なタイプのデータ・ソースを型付けし、カプセル化することができる。図1は、いくつかの例示的タイプのデータ・ソースであり、小フットプリント・デバイス上で動作する典型的なアプリケーション・プログラムのユーザがブックマークを付けることができるものである。

【0015】図1のように、ユーザは特定の電子メールにブックマークを付けることができる。たとえば、ユーザは、電子メールを読んでいる間に、その電子メールのブックマークを作成する特定のコマンドを選択することができる。ユーザは後に、このブックマークを使用してこの電子メールを高速で参照することができる。たとえば、ユーザは、ブックマークのリストを表示するブックマーク・サービス呼び出すことができる。図1は、ブックマークを付けることができる他のデータ・ソースとして、ウェブ・ブラウザ内からブックマークを付けられたウェブ・ページ、および個人情報マネージャ・プログラム内からブックマークを付けられたアポイントメント・エントリなどを示す。

【0016】ブックマークは様々な方法で編成することができる。たとえば、システムは、すべてのブックマークが保存されている中央リストを有することができ、あるいは別々のアプリケーションがそれ自体のブックマーク・リストを有することができ、あるいはこれらの手法

の様々な組み合わせを取ることができる。ブックマークを作成する要求は、ブックマーク・サービス呼び出してブックマークへのデータ・ソースを選択したり、あるいはアプリケーション内からブックマーク・サービス呼び出すコマンドを出すなど、様々な状況で発生する可能性がある。一実施形態では、ブックマークを別のシステムからインポートすることができる。

【0017】図1のブックマークは、ローカルまたはリモートのデータ・ソースを参照することができる。たとえば、図示のウェブ・ページのブックマークは、リモート・サーバに格納されたHTMLページを参照することができ、電子メールのブックマークは、別のシステムに格納された電子メールを参照することができる。

#### 【0018】図2—ブックマークの作成

図2は、特定のデータ・ソースを参照するブックマークを作成するプロセスを示すフローチャートである。図2がブックマーク作成プロセスの一実施形態を示すことに留意されたい。他の実施形態は、様々なステップを追加し、結合し、修正し、あるいは省略することによって可能である。

【0019】図2のステップ600で、ユーザがブックマーク・サービス呼び出す。たとえば、ユーザは、プルダウン・メニュー・コマンドを出してブックマーク・サービス呼び出すことができ、次いでブックマーク・サービスのグラフィカル・ユーザ・インタフェースをディスプレイに表示することができる。ステップ602で、ユーザがブックマークを付ける特定のデータソースを選択する。たとえば、ユーザはディレクトリ構造を走査検索して、仮想名刺、画像など、特定のタイプのデータ・ソースを含むファイルを見つけることができる。ステップ600およびステップ602は、単一のステップに結合することができる。たとえば、ユーザは、アプリケーション内から、現在選択されているデータ・ソースにブックマークを付けるようにブックマーク・サービスに指令するコマンドを出すことができる。

【0020】ステップ604で、ブックマーク・サービスが、ステップ602で選択したデータ・ソースを参照するブックマーク・エントリを作成する。ブックマーク・エントリは、ステップ602で選択した特定のデータ・ソースを識別する情報を含む。一実施形態では、ブックマーク・エントリがURLを含む。

【0021】ステップ606で、ブックマーク・サービスが、ステップ604で作成したブックマーク・エントリを格納する。たとえば、一実施形態では、ブックマーク・エントリを、このブックマーク・サービス自体からアクセス可能であり、かつシステム内で動作している他のアプリケーションからもアクセス可能であるブックマークの中央リストに、格納することができる。別の実施形態では、システムが、異なるアプリケーションまたはデータ・ソース・タイプ毎に別々のブックマーク・リス

10

20

30

40

50

トを保守することができる。

【0022】図3—ブックマークを付けたデータ・ソースの参照

図3は、ブックマークを使用してそれぞれのデータ・ソースを参照するプロセスを示すフローチャートである。図3がブックマークを参照するプロセスの一実施形態を示すことに留意されたい。他の実施形態は、様々なステップを追加し、結合し、修正し、あるいは省略することによって可能である。

【0023】ステップ610で、ユーザがブックマーク・エントリを選択する。ブックマークの選択は、様々な状況で発生する可能性がある。たとえば、ユーザはブックマーク・サービス呼び出し、ブックマーク・サービスによってユーザに提示されたリストから特定のブックマーク・エントリを選択することができる。ユーザは、たとえばブックマーク・メニューから特定のメニュー項目を選択することによって、アプリケーション内から特定のブックマークを選択することもできる。

【0024】ステップ612で、ユーザが、ステップ610で選択したブックマーク・エントリによって参照されたデータ・ソースに対して実行するアクションを選択する。たとえば、ユーザは、ブックマークを付けた電子メールの「編集」アクションや、ブックマークを付けた画像の「表示」アクションなどを選択することができる。ユーザは、実行するアクションを、可能なアクションのリストから選択することによって選択することができる、あるいは、実行するアクションを、ユーザのアクションのコンテキストから暗示的にすることができる。

【0025】ステップ614で、ステップ612で指定したアクションを、ステップ610で選択したブックマーク・エントリによって参照されたデータ・ソースに対して実行するために、適切なプログラム・モジュールが呼び出される。ステップ612および614は、アクションをデータ・ソース・タイプに定義することができ、特定のプログラム・モジュールをこれらのアクションにマッピングすることができる、起動フレームワークが存在することを仮定する。一実施形態では、小フットプリント・デバイス内で動作するブックマーク・サービスおよび/またはアプリケーション・プログラムが、Java Beans™ 起動フレームワーク (JAF) を利用して、ステップ612および614を実装する。Java Beans™ 起動フレームワークについては、以下で説明する。

【0026】ステップ610、612および614は、単一のステップに結合することができる。たとえば、ユーザは、個人用コンタクト・リスト・サービス内から、ブックマークが付いた個人用コンタクト・エントリのリストを表示するメニューにアクセスすることができる。次いで、ユーザは、このメニューからコンタクト・エントリを選択することができ、これによってアプリケーシ

ョンにコンタクト・エントリを表示させることができる。

【0027】Java Beans™ 起動フレームワーク一実施形態では、Java Beans™ 起動フレームワーク (JAF) を利用して、データ・ソースをカプセル化し、データ・ソースにアクションを定義するためのインフラストラクチャを提供する。このセクションでは、Java Beans™ 起動フレームワークを説明する。

【0028】JAFは、いくつかの関連サービスを実装する。これらのサービスには、任意データのタイプを決定すること、データへのアクセスをカプセル化すること、特定のタイプのデータに対して使用可能な操作を発見すること、データの特定部分に対する所望の操作に対応するソフトウェア・コンポーネントをインスタンス化することが含まれる。JAFは、Standard ExtensionとしてJava™ プラットフォームにパッケージ化されている。

【0029】図4は、JAFアーキテクチャを含む主要な要素である。ここに図示したフレームワークは、特定の応用例に結び付けられるものでないことに留意されたい。

【0030】図4のDataHandlerクラスは、JAF認識クライアントと他のサブシステムの間の一貫したインタフェースを提供する。

【0031】DataSourceインタフェースは、データを含むオブジェクトであって、データ・アクセスを提供するストリームとこのデータを記述するMIMEタイプを定義する文字列とを返すことができるオブジェクトをカプセル化する。クラスは、共通のデータ・ソース (ウェブ、ファイル・システム、IMAP、ftpなど) について実装することができる。DataSourceインタフェースは、データ・ソース毎にユーザがカスタマイズできるように拡張することもできる。DataSourceをDataHandlerで設定した後、クライアントがそのデータに対して使用可能な操作を決定することができる。

【0032】JAFは、便宜上、次の2つのDataSourceクラス実装を含む。FileDataSourceは、ファイル内で保持されているデータにアクセスする。URLDataSourceは、URLで保持されているデータにアクセスする。

【0033】CommandMapは、そのインタフェースのコンシューマが、特定のMIMEタイプ上で使用可能な「コマンド」、ならびに特定のMIMEタイプのオブジェクトに対して動作できるオブジェクト (実際にはコンポーネント・レジストリ) を検索するためのインタフェースを決定できるようにするサービスを提供する。CommandMapは、CommandMapの特定のインスタンスの実装によって定義された機構に

よって、特定のデータ・タイプに対して使用可能な機能のリストを生成し、保守することができる。

【0034】JavaBeans™パッケージは、コマンドを実装したソフトウェア・コンポーネントのプログラミング・モデルを提供する。各JavaBeans™コンポーネントは、客観化 (externalization) を使用することができ、あるいは、CommandObjectインタフェースを実装して、型付けされたデータをそれに渡すことができるようにすることができる。

【0035】JAFはCommandMapインタフェースを定義し、これは柔軟性があり拡張可能なCommandMap用フレームワークを提供する。CommandMapインタフェースでは、開発者が、どのコマンドがシステム上で使用可能であるかを発見するための独自のソリューションを開発することができる。可能な実装では、プラットフォーム上の「タイプ・レジストリ」にアクセスでき、あるいはサーバ・ベースのソリューションを使用することができる。JAFは、機能性のような、RFC 1524 (.mailcap) に基づく簡素なデフォルト・ソリューションを提供する。

【0036】Beansは、JAFサービスと対話するためにCommandObjectインタフェースを拡張する。JAFを認識するJavaBeans™コンポーネントは、データ・タイプを検索してそのデータに対して動作するために、それらのDataSourceおよびDataHandlerオブジェクトに直接アクセスすることができる。

【0037】JAFフレームワークの使用

JAFフレームワークの「標準」コンシューマは、DataHandlerを介してこのフレームワークにアクセスする (ただし、主要なサブシステムは独立して動作するようにも設計されている)。下層DataSourceオブジェクトは、DataHandlerクラスが構築されたときに、DataHandlerに関連付けられる。DataHandlerは、データ型付け情報をDataSourceから検索するか、あるいはデータ・タイプを直接コンストラクタから得る。この初期化ステップが完了すると、このデータ項目に対して実行できるコマンドのリストの要求に、DataHandlerからアクセスすることができる。

【0038】アプリケーションがこのリストの要求を出したとき、DataHandlerは、使用可能なコマンドのリストをCommandMapオブジェクトから要求するために、返されるMIMEデータ・タイプのスペシファイアー (specifier) を使用する。CommandMapは、使用可能なコマンド (Beansとして実装される) およびそれらのサポートされているデータ・タイプの知識を有する。CommandMapは、要求されたMIMEタイプおよびCommandMap実装の意味に基づくすべてのコマンドの完全リストのサブ

セットを、DataHandlerに返す。

【0039】最後に、アプリケーションがコマンドのあるデータに適用したいとき、これは適切なDataHandlerインタフェースを通じて実施される。このインタフェースは、CommandMapを使用して、このデータに対して動作するために使用する適切なBeanを検索する。コンテナ (フレームワークのユーザ) が、データとBeanの間の関連付けを行う。

【0040】JAF使用シナリオ

このシナリオでは、JAFを実装しているときに呼び出される標準のタスクの流れを例示するために、仮のファイル・ビューワ・アプリケーションの例を使用する。このファイル・ビューワは、CDEの「dtfile」またはWindows (登録商標) 95 Explorerユーティリティに類似している。このアプリケーションを開始したとき、使用可能なファイルの表示がユーザに提示される。これは、CDEのdtfileまたはExplorerの「右マウス」メニューのような機能を含んでおり、選択されたデータ項目に対して実行できるすべての操作が、その項目のポップアップ・メニューにリストされる。

【0041】一般的なユーザは、このアプリケーションを開始して、ファイルのディレクトリを閲覧する。ユーザがファイルをクリックして指定したとき、このアプリケーションはそのファイルに対して使用可能な操作をリストするポップアップ・メニューを表示する。ファイル・システム・ビューワ・ユーティリティは通常、「編集」、「表示」および「印刷」コマンドを、使用可能な操作として含む。たとえば「表示」を選択すると、このユーティリティは、選択されたファイルを、そのファイルに保持されているデータ・タイプのデータを表示できるビューワ内で開く。

【0042】このアプリケーションによって実行されるタスクの説明を、明瞭にするために次の3つの離散ステップに分類する。

【0043】初期化、すなわち、アプリケーションがファイル・システムの表示を構築する。

【0044】コマンド・リストを得る、すなわち、アプリケーションが、選択されたデータ項目についてのコマンド・リストを提示する。

【0045】コマンドを実行する、すなわち、アプリケーションが、選択されたデータ・オブジェクトに対してコマンドを実行する。

【0046】初期化

以下に述べるインターフェイスの1つは、「DataSource」オブジェクトである。DataSourceオブジェクトは、下層データ・オブジェクトをクラス内にカプセル化する。このクラスは、下層データ格納機構を抽象化し、そのコンシューマに共通のデータ・アクセスおよび型付けインタフェースを提示するものであ

る。ファイル・ビューワ・アプリケーションは、ファイル・システムの内容を照会する。

【0047】ビューワは、ディレクトリ内の各ファイルについてDataSourceオブジェクトをインスタンス化する。次いでビューワは、DataSourceをそのコンストラクタの引数として、DataHandlerをインスタンス化する。DataHandlerは、DataSourceを使わずにインスタンス化することができない。DataHandlerオブジェクトは、クライアント・アプリケーションにCommandMapへのアクセスを提供し、これによって、そのデータに対して動作できるコマンドへのアクセスを可能にするサービスを提供する。このアプリケーションは、DataHandlerオブジェクトのリストを保守し、それらの名前およびアイコンを照会してその表示を生成する。

// ディレクトリ内の各ファイルに対して

File file = new File(file\_name);

DataSource ds = new FileDataSource(file);

DataHandler dh = new DataHandler(ds);

【0048】コマンド・リストを得る

アプリケーションが初期化され、ファイルのリストがユーザーに提示された後、ユーザーはこのリストでファイルを選択することができる。ユーザーがファイルを選択したとき、アプリケーションは、そのファイルに対して使用可能な操作をリストするポップアップ・メニューを表示する。

【0049】アプリケーションは、ファイルに関連付けられたDataHandlerオブジェクトから使用可能なコマンドのリストを要求することによって、この機能性を実装する。DataHandlerは、DataSourceオブジェクトからデータのMIMEタイプを検索し、そのタイプに対して使用可能な操作をCommandMapに照会する。アプリケーションはこのリストを解釈し、これをユーザーにポップアップ・メニュー

// コマンド・オブジェクトを得る

Object cmdBean = cmdInfo[cmd\_id].getCommandObject(dh,  
this.getClassLoader());

... // 適切な場合に直列化／客観化を使用する

my\_aws\_container.add((Component)cmdBean);

【0051】代替シナリオ

第1のシナリオは「標準」の場合であった。アプリケーションがすでにそのデータを表現するオブジェクトを作成済みである状況もある。この場合、既存のオブジェクトをInputStreamに変換したDataSourceのイン・メモリ・インスタンスを作成することは、システム・リソースを非効率的に使用することであり、データの忠実性が損なわれる結果となる可能性がある。

【0052】このような場合、アプリケーションは、D 50

で提示する。次いで、ユーザがこのリストから操作を1つ選択する。

// オブジェクトのコマンド・リストを得る

CommandInfo cmdInfo[] = dh.getPreferredCommands();

PopupMenu popup = new PopupMenu("Item Menu");

// ポップアップに使用可能なコマンドを入れる

for(i = 0; i < cmdInfo.length; i++)

popup.add(cmdInfo[i].getCommandName());

// ポップアップを追加して表示する

add(popup);

popup.show(x\_pos, y\_pos);

【0050】コマンドを実行する

ユーザがポップアップ・メニューからコマンドを選択した後、アプリケーションは適切なCommandInfoクラスを使用して、選択されたコマンドに対応するBeanを検索し、適切な機構(DataHandler、Externalizationなど)を使用してデータをこのBeanに関連付ける。いくつかのCommandObject(たとえばビューワ)はjava.awt.Componentからサブクラス化され、それらが親コンテナを与えられる必要がある。他のCommandObject(デフォルトのprintCommandなど)は、ユーザ・インタフェースを提示しない可能性がある。これによって、スタンド・アロンのビューワ／エディタ、あるいは複合ドキュメント・システム内のコンポーネントとして機能するために十分な柔軟性をそれらに持たせることができる。「アプリケーション」は、CommandObjectが実行するために適当な環境(コンテインメント、ライフ・サイクルなど)を提供する責任を有する。

ataHandler(Object obj, String mimeType) コンストラクタを使用し、DataHandlerをインスタンス化することができる。DataHandlerは、Transferableインタフェースを実装するので、消費側のBeanがInputStream以外の表現を要求することができる。DataHandlerは、要求するコンシューマ用にDataSourceも構築する。DataContentHandler機構は、ObjectからInputStreamへの変換もできるように

拡張される。

【0053】以下のコードは、JAFを使用したデータ

```

/**
 * ビューワに自分の照会結果を表示させる
 */
Component getQueryViewer(QueryObject qo) throws Exception {
    String mime_type = qo.getType();
    Object q_result = qo.getResultObject();
    DataHandler my_dh = new DataHandler(q_result, mime_type);

    return (Component)my_dh.getCommand("view").getCommandObject(my_dh,
null));
}

```

【0054】JAFフレームワーク・コア・クラスインタフェースData Source: DataSourceインタフェースは、Java Beans<sup>TM</sup>起動フレームワークに、ある任意のデータ集合の抽象化を提供する。このインタフェースは、そのデータのタイプならびにそのデータへのアクセスを、InputStreamsおよびOutputStreamsの形式で適切な場合に提供する。

【0055】クラスDataHandler: DataHandlerクラスは、多数の異なるソースおよび形式で使用可能なデータへの一貫したインタフェースを提供する。このクラスは、DataContentHandlersを使用して、文字列変換および関連する操作への単純なストリームを管理する。このクラスは、そのデータに対して動作できるコマンドへのアクセスを提供する。これらのコマンドは、CommandMapを使用して見つけられる。

【0056】インタフェースDataContentHandler: DataContentHandlerインタフェースは、DataHandlerによるTransferableインタフェースの実装を機能拡張するために使用できるオブジェクトによって、実装される。DataContentHandlersを通じて、フレームワークを、ストリームをオブジェクトに変換し、オブジェクトをストリームに書き込むように、拡張することができる。

【0057】インタフェースDataContentHandlerFactory: このインタフェースは、DataContentHandlersのファクトリを定義する。このインタフェースの実装は、MIMEタイプをDataContentHandlerのインスタンスにマッピングするべきである。このインタフェースを実装するクラスの設計パターンは、java.net.URLで使用するContentHandler機構の設計パターンと同じものである。

【0058】クラスCommandMap: CommandMapクラスは、システム内で使用可能なビュー

・ベース・フロント・エンドの一例であり、オブジェクトに関する照会結果を提供するものである。

ワ、エディタ、印刷などのオブジェクトのレジストリへのインタフェースを提供する。開発者は、このパッケージに含まれているCommandMap実装(MailcapCommandMap)を使用するか、あるいは独自のものを開発するかのいずれかを行うことが求められる。このクラスのいくつかのメソッドは抽象であることに留意されたい。

【0059】インタフェースCommandObject: 起動フレームワークを認識するBeansは、このインタフェースを実装して、どのコマンド動詞を実行するように求められているかを見出し、操作すべきデータを表現するDataHandlerを得る。このインタフェースを実装しないBeansも使用することができる。このようなコマンドは、Externalizableインタフェースを使用するか、あるいはアプリケーション固有のメソッドを使用して、データを得ることができる。

【0060】クラスCommandInfo: CommandInfoクラスは、CommandMapの実装によって、コマンド要求の結果を記述するために使用される。このクラスは、要求側に、要求された動詞ならびにbeanのインスタンスを提供する。このコマンドを実装するクラス名を返すメソッドもあるが、それが有効な値を返すことは保証されていない。この理由は、CommandInfoをサブクラス化するCommandMapの実装が、特殊な挙動を提供できるようにするためである。たとえば、FrameworkDeliverablesである。

【0061】JAFフレームワーク補助クラスFileDataSource: FileDataSourceクラスは、ファイルをカプセル化する単純なDataSourceオブジェクトを実装する。このクラスは、FileTypeMapオブジェクトを介してデータ型付けサービスを提供する。

【0062】クラスFileTypeMap: FileTypeMapは、ファイルについてのデータ型付けインタフェースを提供する抽象クラスである。このクラス



の実装は `getContentType` メソッドを実装し、これがコンテンツ・タイプをファイル名または `File` オブジェクトから導出する。`FileTypeMap` は、データ・タイプを決定するために、ファイルのファイル拡張子を検査すること (`MimetypesFileTypeMap` など) から、ファイルを開いてそのタイプをファイルの内容から導出しようと試みることで、いかなるスキームも使用することができる。`FileDataSource` クラスは、デフォルトの `FileTypeMap` (変更されない限り `MimetypesFileTypeMap`) を使用して、ファイルのコンテンツ・タイプを決定する。

【0063】クラス `MimetypesFileTypeMap`: このクラスは `FileTypeMap` を拡張し、ファイルのデータ型付けを、それらのファイル拡張子を介して提供する。このクラスは、`.mime.types` 形式を使用する。

【0064】クラス `URLDataSource:URLDataSource`: `URLDataSource` クラスは、`DataSource` インタフェースで `URL` オブジェクトをラップするオブジェクトを提供する。`URLDataSource` は、`JavaBeans™` 起動フレームワーク内で `URL` によって記述されたデータの処理を簡単にする。これは、このクラスを使用して新規の `DataHandler` を作成することができるからである。

【0065】クラス `MailcapCommandMap:MailcapCommandMap`: `MailcapCommandMap` は、`CommandMap` 抽象クラスを拡張する。このクラスは、その構成が `mailcap` ファイル (RFC 1524) に基づいている `CommandMap` を実装する。`MailcapCommandMap` は、プログラムによっても構成ファイルを介しても構成することができる。

【0066】クラス `ActivationDataFlavor:ActivationDataFlavor`: `ActivationDataFlavor` は、`java.awt.datatransfer.DataFlavor` の特殊なサブクラスである。このクラスによって、`JAF` で、`DataFlavor` クラスによって新規コンストラクタを介して格納された3つの値すべてを設定できるほか、`equals` メソッドでは改良された `MIME` 解析が可能である。この改良された解析を除いて、この意味は `JDK` の `DataFlavor` クラスの意味と同じである。

【0067】クラス `UnsupportedDataTypeException`: 要求された操作が、要求されたデータ・タイプをサポートしていないことを知らせる。

【0068】クラス `MimeType:RFC2045` および `2046` で定義されている、`Multipurpose Internet Extension (MIME)` タイプ。

【0069】小フットプリント・デバイス・アプリケーション/サービス・コンテインメント・フレームワーク  
上述のように、ブックマーク・サービス、または、個人用コンタクト・リスト・サービスなど小フットプリント・デバイス上で実行する別のアプリケーション/サービスは、別々のサービス・モジュールを呼び出して、ユーザが特定のデータ・ソースに選択したアクションを実行することができる。システムをオープン・エンドにすることができ、様々なアクションを様々なタイプのデータ・ソースに定義することができるので、小フットプリント・デバイス・ソフトウェア・アプリケーション/サービスが、モジュール方式で拡張可能な軽量のアプリケーション/サービス・コンテインメント・フレームワークに基づくことが必要であるか、あるいは望ましいことがある。このようなコンテインメント・フレームワークについて、以下に記載する。

【0070】図5—ハードウェア・アーキテクチャのブロック図

図5は、典型的な小フットプリント・デバイスのハードウェア・アーキテクチャを示すブロック図である。本明細書で使用しているように、小フットプリント・デバイスは、プロセッサおよびシステム・メモリなどのコンピューティング・リソースを含むハードウェア・デバイスであるが、これらのリソースの1つまたは複数に、典型的なデスクトップ・コンピュータにある制約よりも著しく大きい制約がある。たとえば、小フットプリント・デバイスは2メガバイト以下のメモリを有することができるが、典型的なデスクトップ・システムは64メガバイト以上のメモリを有することができる。また、典型的な小フットプリント・デバイスは、プロセッサのタイプまたはプロセッサの速度、あるいはその両方のいずれかに関して、典型的なデスクトップ・コンピューティング・システムよりも著しく処理能力が劣るであろう。たとえば、携帯情報端末デバイスは16MHzのプロセッサを有することができるが、典型的なデスクトップ・システムは100MHz以上のプロセッサ速度を有することができる。また、典型的な小フットプリント・デバイスは、デスクトップ・コンピューティング・システムのディスプレイ画面よりも著しく小さいディスプレイ・サイズを有する。たとえば、ハンドヘルド・コンピュータのディスプレイ画面は、一般にデスクトップ・モニタのディスプレイ画面に比較して小さい。

【0071】ここに挙げた特定の数値は例示的なものにすぎず、比較のために使用するものであることに留意されたい。たとえば、8メガバイト以上のメモリを有する携帯情報端末は、なお小フットプリント・デバイスである可能性があるが、このデバイスは、上述の典型的な数字である2メガバイトより多いメモリを有している。

【0072】小フットプリント・デバイスは、上述のメモリ、プロセッサおよびディスプレイ・サイズのリソー

スのほかにも、典型的なデスクトップ・コンピューティング・システムに比較して、他のリソースのタイプに制約があることがある。たとえば、典型的な小フットプリント・デバイスはハード・ディスクを有していないことがあったり、ネットワーク接続を有していないことがあったり、あるいは断続的なネットワーク接続を有したり、あるいは無線ネットワーク接続を有したりすることがある。

【0073】多数の小フットプリント・デバイスは携帯用であり、かつ／またはデスクトップ・コンピュータに比較して小型であるが、必ずしもそうではない。また、多数の小フットプリント・デバイスは、主として、あるいは専ら、バッテリー駆動型でもある。また、小フットプリント・デバイスは典型的に、典型的なデスクトップ・コンピューティング・システムよりも制限された範囲または狭い範囲の使用可能性を有することもある。小フットプリント・デバイスには、これだけに限定されるものではないが、以下の例が含まれる。すなわち、ハンドヘルド・コンピュータ、ウェアラブル・デバイス（たとえば、腕時計コンピュータ）、携帯情報端末（PDA）、

「高性能」携帯電話、セット・トップ・ボックス、ゲーム・コンソール、全世界測位システム（GPS）ユニット、電子テキストブック・デバイスなどである。新しいクラスのコンシューマ・デバイスが急速に出現しているので、小フットプリント・デバイスの網羅的なリストを提供することは不可能である。しかし、「小フットプリント・デバイス」という用語は、上述の用語の精神および範囲内に適度に含めることができるようなデバイスを含むことを意図するものである。

【0074】図5は、典型的な小フットプリント・デバイスのブロック図である。小フットプリント・デバイスが、所望の様々な異なるアーキテクチャを有することができることに留意されたい。本発明の動作を理解する上で必要でないハードウェア要素は、簡素にするため省略した。

【0075】図5のように、小フットプリント・デバイスはプロセッサ100を含む。プロセッサ100は、x86プロセッサを含む様々なタイプのいずれにもすることができる。たとえば、Pentium（登録商標）クラス、PowerPC（登録商標）プロセッサ、ならびに他の低パワーのプロセッサ、または小フットプリント・デバイス専用に開発されたプロセッサである。プロセッサ100は、デスクトップ・コンピュータ・クラスのプロセッサで見られるクロック速度に類似したクロック速度、ならびに16MHzなどの低速を含む、様々なクロック速度を有することができる。

【0076】図5のように、このデバイスはシステム・メモリ102を含む。システム・メモリ102は、RAMまたはROMを含む様々なタイプのメモリを含むことができる。典型的な小フットプリント・デバイスは、典

型的なデスクトップ・コンピュータ・システムに比較して、非常に小さいメモリ記憶容量を有することがある。

【0077】小フットプリント・デバイスは、1つまたは複数の入力機構も含むことができる。入力機構104を図5に示す。入力機構104は、特定のデバイスに適した様々なタイプのいずれにもすることができる。たとえば、入力機構は、キーパッド、マウス、トラックボール、タッチ・ペン、マイクロフォンなどにすることができる。

【0078】小フットプリント・デバイスは、1つまたは複数のディスプレイ機構も含むことができる。ディスプレイ106を図5に示す。しかし、小フットプリント・デバイスはディスプレイを含まないことがあり、あるいはオーディオ・スピーカなど別のタイプの出力機構を含むことがある。ディスプレイ機構106は、特定のデバイスに適した様々なタイプのいずれにもすることができる。高性能携帯電話などの典型的な小フットプリント・デバイス用のディスプレイ機構は、デスクトップ・コンピュータ・システムのディスプレイに比較して小型であることがある。

【0079】図6—ハードウェア／ソフトウェアの階層図

図6は、コンテインメント・フレームワーク内でアプリケーションおよびサービスを実行するシステムに含まれているハードウェア／ソフトウェア層の典型的な階層である。この図は例示的なものであり、特定のデバイスまたは実装に適するように、様々な層を追加し、結合し、あるいは省略することができる。

【0080】図6の基本層がデバイス・ハードウェア層120であり、プロセッサおよびシステム・メモリなど、ソフトウェア・システムをサポートするために必要なハードウェア・リソースを含む。一実施形態では、図5の小フットプリント・デバイス・ハードウェアの例など、小フットプリント・デバイスのハードウェアが、図6のハードウェア層120を実装する。しかし、他の実施形態では、ハードウェア層120は、他のタイプのデバイスで実装することができる。たとえば、スマート・カードなど、典型的な小フットプリント・デバイスよりもさらに大きいリソース上の制約を有するデバイスなどである。

【0081】図6のように、ハードウェア層の次の上層がオペレーティング・システム層122である。当業者には周知であるように、オペレーティング・システムは、デバイス・ハードウェアと、そのデバイス上で動作するソフトウェアの間のインタフェース層として機能し、入出力、メモリ管理などの低レベルのタスクのマネージャとしての機能を果たす。図6のオペレーティング・システム122は、図6のより高い層をサポートするいかなる特定のオペレーティング・システムにもすることができる。オペレーティング・システム122は、小

フットプリント・デバイスで使用するために適しているか、あるいはそのために特に書かれた、小型で効率的なものにすることができる。たとえば、オペレーティング・システム 122 は、Sun Microsystems 社から入手可能な JavaOS オペレーティング・システムにすることができる。

【0082】一実施形態では、コンテナメント・フレームワークが、Java™ アプリケーション環境において 1 つまたは複数の Java™ クラスとして実装される。図 6 のように、Java™ 仮想マシン層 124 および Java™ アプリケーション・プログラミング・インタフェース (API) クラス・ライブラリ層 126 は、オペレーティング・システムの次の上層である。これらの 2 層が共に Java™ アプリケーション環境または Java™ プラットフォームを作成する。コンテナメント・フレームワークを実装するクラスは、Java™ ライブラリ 126 を使用して構築することができ、バイトコードにコンパイルすることができる。このバイトコードは、Java™ 仮想マシン 124 上で実行する命令であり、オペレーティング・システム 122 および/またはデバイス・ハードウェア 120 と対話する。

【0083】一実施形態では、コンテナメント・フレームワークが Personal Java Java™ アプリケーション環境で実装される。これは、高スケーラブルでモジュール方式、かつ構成可能に設計された Java™ プラットフォームであり、最低限のシステム・リソースを必要とするものである。Personal Java™ は、Java™ 仮想マシンおよび Java™ API のサブセットを含み、コアおよびオプションの API およびクラス・ライブラリを含む。さらに、Personal Java™ API は、専用バージョンの Java™ abstract window toolkit (AWT) など、リソースが制限された環境でコンシューマ・アプリケーションが要求する特定の機能を含む。Personal Java™ AWT ライブラリは、コンシューマ製品のルック・アンド・フィールを目標として調整されており、グラフィックスおよびウィンドウ機能を提供し、低解像度のディスプレイおよび代替入力デバイスをサポートする (マウスおよびキーボードのないデバイス用に拡張されたイベント・モデルを介する)。

【0084】再度図 6 を参照して、コンテナメント・フレームワーク 128 を Java™ プラットフォーム層の次の上層として示す。上述のように、コンテナメント・フレームワーク 128 を他のプラットフォームに基づかせることもできる。以下で詳細に述べるように、コンテナメント・フレームワーク 128 は、たとえばモジュール登録、ルックアップ、インスタンス追跡などを可能にすることによって、プログラム・モジュールを管理する。モジュールは様々なサービスを提供することが

できる。コンテナメント・フレームワーク 128 は、モジュールが他のモジュールを要求してそれらのサービスを使用できるようにする。アプリケーションは、他のモジュールのサービスを利用するモジュールとして実装することができる。したがって、コンテナメント・フレームワーク 128 は、軽量で拡張可能なサービスおよびアプリケーション・フレームワークを提供し、アプリケーションが共存してモジュラー・コード・ベースを共有できるようにする。

【0085】このタイプの拡張可能なアーキテクチャは、多数のプログラム・モジュールを協調可能にするものであり、小フットプリント・デバイスには重要な開発である。小フットプリント・デバイスは、歴史的に、相対的に狭い用途に制限されてきた。たとえば、携帯電話は通常、テレフォニーおよび他のわずかな用途に使用されていた。しかし、様々な技術が開発され、小フットプリント・デバイスが「より高性能」になり、汎用プロセッサ、より大型のディスプレイ画面などを有することが可能になるので、小フットプリント・デバイスで利用されるアプリケーションの範囲を拡張することが望ましくなっている。

【0086】このコンテナメント・フレームワークは、一般にデスクトップ・コンピューティング環境に関連するタイプのアプリケーションおよびサービスが、小フットプリント・デバイスにおいて、デスクトップ・コンピュータのユーザが慣れている様式で、共に動作できるようにすることができる。図 6 および上述のように、小フットプリント・デバイス上で動作するサービスおよびアプリケーション 130 は、コンテナメント・フレームワーク層 128 上で構築されたモジュールとして実装することができる。たとえば、Sun Microsystems 社から入手可能な Personal Application Suite は、コンテナメント・フレームワーク 128 の一実施形態を用いて構築されている。Personal Application Suite は、ブラウザ、電子メール・クライアント、およびパーソナル・オーガナイザなどのアプリケーションの統合セットを含んでいる。

【0087】図 6 は、コンテナメント・フレームワーク 128 のいくつかの実施形態の、オフ・デバイス・サービス 132 をオン・デバイス・アプリケーション/サービス 130 と統合する能力も示す。たとえば、コンテナメント・フレームワーク 128 は、小フットプリント・デバイスと Jini ネットワークなどのネットワークの間のインタフェースを提供することができる。小フットプリント・デバイス・システムは、そのサービスを、ネットワーク内の他のデバイスまたはクライアントが使用するように登録することができる。コンテナメント・フレームワークは、小フットプリント・デバイス内のサービスおよびアプリケーションが、他のネットワ

ーク・デバイスが提供するサービスをルック・アップおよび使用できるようにすることもできる。小フットプリント・デバイスのサービスをネットワーク・サービスと統合することについては、以下で図7についてより詳細に論ずる。

【0088】図7-11：例示的ネットワーク・デバイスおよびサービス・フェデレーション

図7は例示的ネットワークであり、コンテインメント・フレームワーク内のアプリケーション／サービスを実行する小フットプリント・デバイスが、ローカル・サービス・ベースのネットワークに接続される。図示の例では、コンテインメント・フレームワーク144を利用する高性能携帯電話134が、ネットワークに接続される。図ではまた、ネットワークに、プリンタ130およびインターネット使用可能テレビ132も接続されている。この例では、プリンタ130およびテレビ132のデバイスが、サービスをネットワークにエクスポートし、場合によってはネットワーク上の他のデバイスのサービスを使用するように、動作可能であると仮定する。たとえば、プリンタがそのプリント・サービス138をエクスポートすることができ、インターネット・テレビがそのプリント・サービスをルックアップして使用し、ウェブ・ページを印刷することができる。デバイスおよびサービスのフェデレーションをこの方法で容易にするために、ルックアップ・サービス136はネットワーク上に位置している。ルックアップ・サービス136は、ネットワーク・サーバなどの別のデバイス上に存在することができる。

【0089】デバイスおよびサービスのフェデレーションは、様々な方法で実装することができる。たとえば、Sun Microsystems社から入手可能なJini<sup>TM</sup>技術は、図7のタイプの分散システムを使用可能にするコンポーネントおよびプログラミング・モデルを含む。一実施形態では、図7のローカル・ネットワークをJini<sup>TM</sup>ネットワークにすることができ、プリンタ130およびインターネット・テレビ132をJini<sup>TM</sup>使用可能デバイスにすることができる。各デバイスは、Jini<sup>TM</sup>ネットワーク・ルックアップ・サービスを見つけて、そのデバイスが提供するサービスをこのルックアップ・サービスに登録するように、動作可能である。このルックアップ・サービスは、サービスが提供する機能性を示すインタフェースを、そのサービスを実装するオブジェクトのセットにマッピングする。

【0090】デバイスまたは他のサービス・プロバイダは、そのサービスをサービス・フェデレーションへ追加するために、まず「発見」プロトコルを使用して適切なルックアップ・サービスを探し出すことができる。図8は、発見プロセスである。図示のように、サービス・プロバイダ164、たとえば図7のプリンタ130は、ローカル・ネットワーク上で、ルックアップ・サービスに

それら自体を識別させる要求を同報通信することができる。

【0091】サービス・プロバイダ164がルックアップ・サービス160を探し出した後、サービス・プロバイダ164は、「結合」プロトコルを使用して、そのサービスをルックアップ・サービス160に登録することができる。図9は、結合プロセスである。サービス・プロバイダ164は、クライアントがそのサービスを呼び出すために使用できるサービス・オブジェクトを作成することができる。図9のように、次いで、提供されたサービスのサービス・オブジェクトを、提供されたサービスのタイプや名前についての情報を含むサービス属性または記述子と共に、ルックアップ・サービス160にロードすることができる。たとえば、Jini<sup>TM</sup>ネットワークでは、図7のプリンタ130が、プリント・サービス138用のJava<sup>TM</sup>プログラミング・インタフェースを含むサービス・オブジェクトを作成することができる。次いで、プリンタ130は、ルックアップ・サービス136の「登録」メソッドを呼び出して、このサービス・オブジェクトを、登録中のサービス138がプリント・サービス、印刷解像度、可能な用紙サイズなどであると指定する属性と共に、渡すことができる。

【0092】サービス・プロバイダ164がそのサービスをルックアップ・サービス160に結合した後、他のネットワーク・クライアントがこのサービスを要求し、使用することができる。サービスを要求するプロセスはルックアップと呼ばれ、これを図10に示す。ルックアップ・サービスを発見した後、クライアント162は、要求するサービスの記述を使用して、ルックアップ・サービス160からサービスを要求することができる。ルックアップ・サービス160は、要求側から与えられた記述を、ルックアップ・サービスに結合されたサービスに突き合わせようと試みる。ルックアップ・サービス160は、結合プロセス中にサービス・プロバイダ164から送信されたサービス属性を使用して、この突き合わせを実行することができる。合致が発見された場合、ルックアップ・サービス160は適切なサービス・オブジェクトをクライアント162へ提供する。たとえば、要求されたサービス用のJava<sup>TM</sup>インタフェースを、クライアント162へ提供することができる。

【0093】クライアント162がサービス・オブジェクトをルックアップ・サービスから受け取った後、クライアントがこのサービスを呼び出すことができる。図11は、サービスを呼び出すプロセスである。サービスが呼び出されたとき、クライアント162およびサービス・プロバイダ164は直接互いに通信することができる。様々な対話プロトコルのいずれも、この通信に使用することができる。たとえば、使用するプロトコルを、Java<sup>TM</sup> Remote Method Invocation (RMI)、CORBA、DCOMなどにす

ることができる。クライアントがルックアップ・サービスから受け取るサービス・オブジェクトは、たとえば RMI メソッドを呼び出すことによって、サービス・プロバイダに位置するコードヘコール・バックすることができ、あるいは、要求されたサービスを提供するようにローカルで実行することができ、あるいは、これらの手法を組み合わせ使用することができる。

【0094】図7のように、ローカル・ネットワーク用のルックアップ・サービス 136 は、インターネット 154 など外部ネットワークへのゲートウェイとしても動作することができる。したがって、サービス・ベースの分散コンピューティング・モデルを、ローカル・ネットワーク外部に位置するクライアントおよびサービスを含めるように拡張することができる。たとえば、Open Service Gateway Initiative (OSGI) 用に開発されている技術を利用して、このタイプの分散コンピューティング・システムを実装することができる。

【0095】このタイプの、異なるネットワークとインターネットの間のサービス共有によって、新しいタイプのアプリケーションを開発可能にすることができる。たとえば、商人はインターネット・サービスを使用して特定の顧客についてのデータを記録することができ、広告サービス・プロバイダはこのデータを使用して、どのローカル・ネットワークにコンシューマ・デバイスが接続されているかなどに応じて、コンテキスト特異的な広告をコンシューマ・デバイスに出すことができる。たとえば、顧客はショッピング・モールに入り、無線接続を介して携帯情報端末 (PDA) をこのショッピング・モールのローカル・ネットワークに接続することができる。インターネット・ベースの顧客データ・サービスをショッピング・モール・ネットワーク用のルックアップ・サービスに結合することができ、このモール・ネットワークに接続したばかりの特定の顧客についての情報を提供することができる。次いで、このショッピング・モール・ネットワーク内で動作するサービスは、このデータを、モール内の顧客の現在位置、時間などの他の要素と共に使用して、個別化された広告を生成して顧客の PDA に出すことができる。

【0096】図7のネットワークに基づく他の多数のサービス例が可能である。たとえば、住宅内のネットワーク使用可能コンシューマ・デバイスは、電力会社が提供する、住宅内の電力消費量を管理するサービスを、インターネットを介して利用することができ、セキュリティ・サービス・プロバイダは、ネットワーク・サービスを介して家庭または特定のデバイスを監視することができ、資産が荒らされたときに直ちに所有者に通知することができ、健康サービス・プロバイダは、医療機器と通信することによって患者の容態をリモートで監視することができる。

【0097】上に挙げた例では、デバイスが透過的にネットワークに接続することができ、ネットワーク・サービスをデバイス常駐サービスと統合することができ、ネットワーク・クライアントが使用するためにデバイス常駐サービスをエクスポートすることができると仮定する。本明細書で説明するコンテインメント・フレームワークは、携帯情報端末、ハンドヘルド・コンピュータ、高性能携帯電話などの小フットプリント・デバイスのサービスおよびアプリケーションを、ネットワーク・サービス・フェデレーションと統合するために必要なインタフェースを提供することができる。

【0098】図7に図示し、以下により詳細に説明するように、コンテインメント・フレームワーク 144 は、所有するタイプのルックアップ・サービス 146 を有する。コンテインメント・フレームワーク 144 内のルックアップ・サービス 146 は、上述のローカル・ネットワーク・ルックアップ・サービスと類似した方法で動作することができ、発見、結合、ルックアップ、およびサービス呼び出しプロセスを利用する。たとえば、パーソナル・オーガナイザ・アプリケーション 152 は、カレンダー・サービス、コンタクト・リスト・サービス、ブックマーク・サービスなど、様々なサービスを利用することができる (図示せず)。パーソナル・オーガナイザ・アプリケーション 152 は、これらのサービスと通信するための参照を、コンテインメント・フレームワーク・ルックアップ・サービス 146 を介して得ることができる。

【0099】コンテインメント・フレームワーク 144 は、所有するルックアップ・サービス 146 を、図7のローカル・ネットワーク・ルックアップ・サービス 136 などのオフ・デバイス・ルックアップ・サービスに統合することができる。このように、プリント・サービス 138 およびウェブ・サービス 140 などのオフ・デバイス・サービスを、コンテインメント・フレームワークのアプリケーション/サービス 148、150 および 152 で使用可能にすることができ、その逆に使用することもできる。たとえば、パーソナル・オーガナイザ・アプリケーション 152 は、プリント・サービスをコンテインメント・フレームワーク・ルックアップ・サービス 146 から要求することができる。コンテインメント・フレームワーク・ルックアップ・サービス 146 はまず、オン・デバイスプリント・サービスを探索することができる。これが見つからなかった場合、コンテインメント・フレームワーク・ルックアップ・サービス 146 は、プリント・サービスをネットワーク・ルックアップ・サービス 136 から要求することができる。次いで、プリント・サービス 138 のサービス・オブジェクトをパーソナル・オーガナイザ 152 へ返すことができる。オン・デバイス・サービス/アプリケーションとオフ・デバイス・サービスの間のインタフェース 142 を、図

7に示す。オン・デバイス／オフ・デバイス・サービスの統合を実装する方法について、以下に詳述する。

【0100】上述のように、サービスのクライアントはそれ自体を他のクライアントへのサービスにすることができる。たとえば、図7の高性能携帯電話の電子メール・クライアント「アプリケーション」150は、それ自体を、コンテインメント・フレームワーク144で動作するクライアントまたはネットワーク・クライアントへのサービスにすることができる。たとえば、誤動作が起こった場合、図7のプリンタ130が電子メール・サービスを要求して、診断情報をサービス技術者へ送信できるようにすることができる。ネットワーク・ルックアップ・サービス136がネットワーク・ベースの電子メール・サービスを見つけることができなかつた場合、インタフェース142を介して電子メール・サービスを高性能携帯電話134から要求することができる。コンテインメント・フレームワーク144で動作する電子メール・アプリケーション／サービス150のサービス・オブジェクトを、要求側のプリンタ・クライアント130へ渡すことができる。この例では、プリンタ・クライアント130が直接電子メール・アプリケーション／サービス150と通信して、診断情報を含む電子メールをプリンタ・サービス技術者へ送信することができる。電子メール・アプリケーション／サービス150は、電子メール・サーバ・サービスを見つけることができた場合に電子メールを直ちに送信することができ、あるいは、後に携帯電話のユーザが異なるネットワークへ接続する際にこのようなサービスが使用可能になるときに、この電子メールを送信することができる。

【0101】上記の説明は、Jini<sup>TM</sup>技術など、特定のプロトコルおよびプログラミング・モデルを参照するが、これらの特定の技術は例示的なものにすぎないことに留意されたい。たとえば、コンテインメント・フレームワーク内のアプリケーションおよびサービスを、様々なタイプの規格、プロトコル、およびプログラミング・モデルを使用するクライアント、サービス、デバイス、ネットワークなどと統合することができる。これらの規格、プロトコル、およびプログラミング・モデルには、Jini<sup>TM</sup>、CORBA、COM/DCOM、Bluetooth、CAL、CEBus、HAVi、Home API、HomePNA、HomePnP、HomeRF、VESAなどが含まれるが、それだけに限定されるものではない。

【0102】図12—コンテインメント・フレームワークのブロック図

図12は、コンテインメント・フレームワーク環境の基本アーキテクチャを示す抽象的なブロック図である。上述のように、コンテインメント・フレームワークはアプリケーションおよびサービス用のコンテインメント・システムを提供する。これらのアプリケーションおよびサ

ービスは、システム内でモジュールと呼ばれるユニットとして管理される。コンテインメント・フレームワークは軽量であり、一実施形態では、モジュールが、すべてのモジュール管理を実行する単一のフレームワーク・マネージャと対話することができる。このマネージャを、本明細書では中央フレームワーク・インスタンスと呼ぶ。一実施形態では、中央フレームワーク・インスタンスをJava<sup>TM</sup>クラスのインスタンスとして実装することができる。図12は、中央フレームワーク・インスタンス170、およびそれが含む／管理するコードおよびデータである。図12がコンテインメント・フレームワークの一実施形態を示すことに留意されたい。他の実施形態は、異なるアーキテクチャを使用することができ、かつ／または異なるプログラミング言語またはソフトウェア環境で実装することができる。たとえば、図12の中央フレームワーク・インスタンス170が実行するモジュール管理／コンテインメントを、他の実施形態では、多数のオブジェクトまたはコンポーネントが実行することができる。

【0103】図12のように、中央フレームワーク・インスタンス170は、現在システムにロードされているモジュールを表現するデータ182を含む。コンテインメント・フレームワーク・アーキテクチャは非階層である。したがって、ロードされたモジュールを、モジュールの単層リストまたは配列として表現することができる。この非階層システムは、コア・コンテインメント・フレームワーク・コードおよびフレームワーク内で動作するモジュールを小さく保つための助けとなる。Java Beans<sup>TM</sup>コンポーネントなど、階層コンポーネントを使用するシステムは、それに伴う利点を提供することができるが、これらの利点は、より多くのシステム・リソースを必要とするより複雑な管理システムを負担することで得られる。しかし、コンテインメント・フレームワークは、非階層モジュールが階層コンテインメント・システムの利点を多数得るための機構を提供する。この機構を、図13および図14について以下に説明する。

【0104】図12のように、一実施形態では、中央フレームワーク・インスタンス170が、公開されたアクセス可能なメソッド178を含み、モジュールがこれら呼び出すことができる。これらのメソッドを抽象グループに分割することができる。たとえば、あるグループのメソッド172には、ルックアップ・メソッドを含めることができる。ルックアップ・メソッドは、上述のルックアップ・サービス機能性を実装する。モジュールは、モジュール記述子を中央フレームワーク・インスタンス170のルックアップ・メソッドに渡して、特定のサービス・モジュールを探し出すことができる。コンテインメント・フレームワーク・ルックアップ・プロセスについては、以下で図16について説明する。別のグル

ープのフレームワーク・メソッド174には、モジュールをロードおよびアンロードするメソッドを含めることができる。サービス・モジュールを見つけた後、クライアント・モジュールは中央フレームワーク・インスタンス170に対して、このサービス・モジュールをロードして、ロードしたモジュールへの参照を返すように要求することができる。次いで、クライアント・モジュールがこのサービスを読み出すことができる。クライアントは、サービス・モジュールを使用し終えたときに、フレームワーク・メソッドを読み出してこのサービス・モジュールを解放することができる。異なるグループとして記載したが、メソッドをルックアップおよびロード／アンロードのグループに分割することは、概念上の分割にすぎないことがある。たとえば、一実施形態では、ルックアップ・メソッドが、合致するモジュールをロードして、合致したモジュールへの参照を返すこともできる。

【0105】図12は、フレームワーク・メタデータと呼ばれるシステム・データ180も示す。これは、ロードしたモジュールのリストを記述するデータ182およびシステムの状態を記述する他のデータを含むことができる。中央フレームワーク・インスタンス170の別の抽象グループのメソッド176には、リフレクション・メソッドを含めることができる。リフレクション・メソッドは、コア・メタデータ180への直接アクセスを提供するので、他のグループのメソッドとはやや異なる。システム・モジュールと呼ばれるモジュールの特殊クラスが、リフレクション・メソッドを読み出して、メタデータ180へのアクセスを得ることができる。通常のモジュールはメタデータ180にアクセスすることができない。

【0106】コア・システム・データ180への参照を受け取った後、システム・モジュールはいかなる望ましい方法でもデータを使用あるいは修正することができる。したがって、コンテインメント・フレームワークは大幅に拡張可能である。中央フレームワーク・インスタンス170はそれ自体を小さく保つことができ、システム・モジュールを追加して、中央フレームワーク・インスタンス170によってまだ使用可能にされていない機能性を実装することができる。たとえば、システム・モジュールは、図7から図11について上述した、コンテインメント・フレームワーク内で動作するアプリケーション／サービスと外部ネットワークに基づくサービスの間の統合を、可能にすることができる。

【0107】この例では、このようなシステム・モジュールを、外部ネットワークのプロトコルおよびプログラミング・モデルに従う第2のルックアップ・サービスとして書くことができる。たとえば、Jini<sup>TM</sup>ネットワークでは、Jini<sup>TM</sup>ネットワーク・ルックアップ・サービスを発見してネットワーク・ルックアップ・サービスを結合し、それ自体を第2のルックアップ・サービス

として登録するシステム・モジュールを書くことができる。ネットワーク・クライアントがサービスを要求したとき、ネットワーク・ルックアップ・サービスが、システム・モジュールによって実装されたルックアップ・サービスを呼び出すことができる。このシステム・モジュールは、コンテインメント・フレームワーク内で、要求したサービスの記述に合致するサービス・モジュールを発見しようと試みることができる。合致が見つかった場合、システム・モジュールは、このサービス・モジュールをネットワーク・クライアントへエクスポートするために必要なステップを実行することができる。これは、システム・モジュールがシステム・モジュール・リストおよびメタデータへの完全アクセスを有しているからである。たとえば、システム・モジュールは、合致したサービス・モジュールをシステムへロードして登録し、Java<sup>TM</sup>インタフェースなど、新たにロードしたモジュールへのインタフェースを要求側に返すことができる。

【0108】図13および図14—階層環境のシミュレート

20 モジュールの階層コンテキストを確立することは、しばしば望ましい。たとえば、同じタイプのいくつかのサービス・モジュールがシステム内に存在することができるが、それぞれがわずかに異なる挙動を示すことができる。階層コンテインメント・システムでは、モジュールによるサービスの要求を、要求側モジュールの親または含んでいるモジュールを通じてフィルタリングすることができ、特定のサービス・モジュールへの参照を要求側へ戻すことができるようにする。階層コンテインメントは、他の固有の利点も有する。たとえば、データをモジュールの階層の間で容易に分散させ、格納する能力などである。しかし、上述のように、階層コンテインメント・システムの完全実装には、メモリおよび処理能力など、必要なシステム・リソースに関して非常に費用がかかることがある。コンテインメント・フレームワークは、開発者とアプリケーションに、階層コンテインメントの利点を多数もたらす機構を提供することができるが、通常、これに伴う高い間接経費はかからない。

30 【0109】たとえば、一実施形態のコンテインメント・フレームワークでは、モジュールがそれ自体を、他のモジュールのモジュール要求リスナとして登録することができる。たとえば、モジュールAは、それ自体を、モジュールBの要求リスナとして登録することができる。これは、たとえば、中央フレームワーク・インスタンスのAddRequestListenerメソッドを読み出すことによって行う。続いて、モジュールBが、特定のサービスを見つけるために中央フレームワーク・インスタンスのメソッドを読み出したとき、中央フレームワーク・インスタンスは、モジュールBのモジュール要求リスナがあるかどうかを検査する。この場合、中央フレームワーク・インスタンスはモジュールAを要求リス

ナとして見つけ、モジュールAに対して、要求されたサービス・モジュールをモジュールBへ提供するように求める。

【0110】図13および図14は、コンテインメント・フレームワークにおけるモジュール要求リスナの例示的使用である。図13は、プリント・サービスの所望の概念上のモジュール階層である。図示のように、2つのプリント・サービス・モジュール192および194、プリント・サービスAおよびプリント・サービスBが、プリント・マネージャ・モジュール190内にカプセル化されている。たとえば、2つのプリント・サービス192および194は、異なる場所に印刷することができ、異なる解像度および色機能などを有することができる。これらのプリント・サービス・モジュールのいずれかが、別のモジュールによるプリント・サービスのルックアップ要求を満たすことができる。しかし、特定のプリント・サービスを選択して返すプリント・マネージャ・モジュールを使用することが望ましいことがある。たとえば、プリント・マネージャ190は、どのクライアント・モジュールが印刷要求を行うかに基づいてプリント・サービスを選択することができ、あるいは、プリント・マネージャは、ユーザに対して所望のプリント・サービス特性を入力するように求めるダイアログ・ボックスを表示することができる。

【0111】コンテインメント・フレームワークは非階層コンテインメント・モデルを利用するが、図13の階層を実現することができる。これは、プリント・マネージャ・モジュール190を、プリント・サービスを要求することができるクライアント・モジュールのモジュール要求リスナとして登録することによって行う。図14は、システム内で動作することができるモジュール例198である。先に述べたように、これらのモジュールはみずから他のモジュールをサービスとして使用することができる。コンテインメント・フレームワークの非階層モデルに従って、モジュールを、単層のレイアウトで固有のモジュール階層を使わずに配列して示す。

【0112】この例では、ウェブ・ブラウザ・モジュール196が、たとえばウェブ・ページを印刷するために、印刷要求を行うように動作可能にすることができる。図14のように、プリント・マネージャ・モジュール190を、ウェブ・ブラウザ・モジュール196のモジュール要求リスナとして登録することができる。プリント・サービス要求をウェブ・ブラウザ196から受け取ると、コンテインメント・フレームワーク・ルックアップ・サービスは、ウェブ・ブラウザ・モジュール196の要求リスナとして登録されたプリント・マネージャ・モジュール190を見つけることができ、プリント・マネージャ・モジュール190に対して、プリント・サービス・モジュールをウェブ・ブラウザ要求例196へ提供するように求めることができる。次いで、プリント

・マネージャ・モジュール190が、プリント・サービス・モジュールA192またはプリント・サービス・モジュールB194への参照を返すことができ、あるいは、プリント・マネージャ・モジュール190が、ダイアログ・ボックスをユーザに提示して、どのプリント・サービス・モジュールを返すかを判断することができる。したがって、図13の所望のモジュール階層を、コンテインメント・フレームワークの非階層モジュールのために実装することができる。

【0113】図15—パーセル・パッケージ化ユニットモジュールを、パーセルと呼ばれるユニットにパッケージ化することができる。このパッケージ化はいくつかの目的に役立つ。たとえば、パーセルは、関連するコードおよびデータをユニットとして管理するために好都合な機構を提供する。密接に関連したモジュールが静的な従属性を有する場合、これらを共にパーセルにパッケージ化することができる。パーセルを使用して、システム内のインストレーションおよびアップグレードを処理することができる。

【0114】図15は、個人情報マネージャ（PIM）に関するモジュールを共にグループ化するパーセル例200である。この図では、カレンダー・モジュール202、コンタクト・リスト・モジュール204、アポイントメント・モジュール208、およびユーザ・インタフェース・モジュール206を示す。他の様々なモジュールを、所望するようにパーセル内に存在させることができる。PIMパーセル200のモジュールは、ブックマーク・サービス、検索サービスなど、コンテインメント・フレームワーク内で動作する様々なコア・サービス・モジュールを使用することもできる。PIMパーセルを使用すると、PIMアプリケーションのインストレーションおよびアップグレードを簡素化することができる。PIMモジュールをこのようにしてパーセルにパッケージ化すると、多目的な開発のために別々のコード・ユニットを作成する際の開発時間の利点もある。

【0115】パーセルは、非階層モジュールのランタイム・コンテキストを提供するための追加の方法も提供する。モジュールがシステムにロードされたとき、中央フレームワーク・インスタンスは、このモジュールが属するパーセルがあればそれを指定するメタデータを格納することができる。サービス・モジュールは後にこの情報を使用して、異なるクライアント・モジュールに対し、そのクライアントが属するパーセルに応じて異なる方法でサービスを提供することができる。たとえば、クライアント・モジュールは、ファイル・アクセス・サービス・モジュールを使用してルート・ディレクトリを得ることができる。ファイル・アクセス・モジュールは、異なるクライアントに対し、そのクライアントが属するパーセルに応じて異なるルート・ディレクトリを返すことができる。



【0116】図16-モジュール要求フローチャート  
図16は、典型的なルックアップ・プロセスを示すフローチャートであり、このプロセスでは、中央フレームワーク・インスタンスが、サービス・モジュールへのルックアップ要求をクライアント・モジュールから受け取ったときに実行することができる。図16が例示的なものであり、様々なステップを結合し、省略し、あるいは修正することができることに留意されたい。たとえば、先に述べたように、ルックアップ・プロセスをカスタマイズするシステム・モジュールを追加することができる。

【0117】図16のステップ300で、中央フレームワーク・インスタンスがモジュール・ルックアップ要求を要求側モジュールから受け取る。たとえば、要求側モジュールは、中央フレームワーク・インスタンスの `RequestModule` メソッドを呼び出して、要求されているサービス・モジュールのモジュール記述子、ならびに要求側モジュール自体への参照を渡すことができる。要求側モジュールへの参照をシステム・データに追加して、サービス・モジュール・ユーザの追跡を維持することができる。以下でより詳細に述べるように、他のモジュールがモジュールを使用していないときは、このモジュールをアンロードすることができる。

【0118】要求側モジュールによって渡されたモジュール記述子は、要求したモジュールについての様々な属性を指定し、フレームワーク・インスタンスがこの属性を使用して、合致するモジュールを見つけようと試みることができる。このモジュール記述子は、要求したモジュールのサービス・タイプ、クラス名、および/またはサービス固有の属性などの情報を含むオブジェクトにすることができる。要求側はテキスト記述を中央フレームワーク・インスタンスに渡すこともでき、中央フレームワーク・インスタンスはこの記述を使用して、モジュール記述子オブジェクトを作成することができる。

【0119】ステップ302で、中央フレームワーク・インスタンスは、要求リスナ・モジュールが要求モジュールについて登録されているかどうかを検査する。要求リスナが見つかった場合、ステップ304で、フレームワーク・インスタンスが要求リスナにこの要求を通知し、要求リスナに対して、モジュール要求記述子に合致するモジュールを提供しようと試みるように指令する。要求リスナが合致するモジュールを提供できた場合、実行はステップ314へ進行する。そうでない場合は、登録されている他の要求リスナに対して、合致が見つかるかあるいはそれ以上の要求リスナがなくなるまで、モジュールを提供するように求めることができる。

【0120】要求リスナが見つからなかった場合、あるいはどの要求リスナも要求されたモジュールを提供できなかった場合、実行はステップ306へ進行する。しかし、一実施形態では、1つまたは複数の要求リスナが要求モジュールについて登録されており、そのうちのい

れも合致するモジュールを提供できなかった場合、実行はステップ304の後で停止することがある。ステップ306で、中央フレームワーク・インスタンスがモジュールのリストを検査して、モジュールの1つがモジュール記述子に合致するかどうかを判定する。合致が見つかった場合、ステップ308で、フレームワーク・インスタンスが、合致したモジュールがマルチ・インスタンス化可能かどうかを検査する。そうでない場合、実行はステップ314へ進行する。

【0121】ステップ308で、合致したモジュールがマルチ・インスタンス化可能であることが分かった場合、中央フレームワーク・インスタンスは合致を求めてモジュール・リストの探索を継続することができる。探索するモジュールがそれ以上なくなった場合、実行はステップ310へ進行する。ステップ310で、フレームワーク・インスタンスがモジュール提供側モジュールをモジュール・リスト内で探索する。モジュール提供側モジュールは、要求されたモジュールを提供することができるモジュールである。たとえば、ネットワーク・ルックアップ・サービスを、コンテインメント・フレームワーク用のモジュール提供側モジュールとしてインポートすることができる。

【0122】モジュール提供側モジュールが見つかった場合、ステップ312で、中央フレームワーク・インスタンスがモジュール提供側モジュールに要求を通知し、モジュール要求記述子に合致するモジュールを提供しようと試みるように指令する。合致が見つかった場合、実行はステップ314へ進行する。モジュール提供側が要求されたモジュールを提供できなかった場合、中央フレームワーク・インスタンスは他のモジュール提供側モジュールを探索して、ステップ312を繰り返すことができる。モジュール提供側がモジュール・リスト内に存在しない場合、あるいはいずれも要求されたモジュールを提供できなかった場合、要求側に、この要求を満たすことができないことが通知され、実行が完了する。

【0123】ステップ314には、ステップ304、308または312から到達することができる。すべての場合において、モジュール要求記述子に合致するモジュールが見つけられる。ステップ314で、要求側は合致したモジュールのユーザとして登録され、ステップ316で、合致したモジュールへの参照が要求側に返される。合致したモジュールのロードおよび初期化に含まれる、いかなる必要な初期化ステップも、ステップ314で実行される。たとえば、モジュールは、モジュールがロードされたときに呼び出される `Initialize` メソッドを有することができる。

【0124】上述のように、図16のフローチャートは例示的なものであり、様々な実施形態が異なるルックアップ/ロード・シナリオを有することができる。たとえば、モジュールは、中央フレームワーク・メソッドを呼

10

20

30

40

50

び出して、合致したモジュールへの参照を返すことなくサービス・モジュールをロードすることができ、あるいはいくつかの場合で要求リスナを無視することができる。

【0125】図17—モジュール解放フローチャート  
クライアント・モジュールがサービス・モジュールを使用し終えたとき、クライアントは中央フレームワーク・インスタンスのメソッドを呼び出して、このモジュールを解放することができる。図17は、モジュール解放プロセスを示すフローチャートである。図17のフローチャートは例示的なものであり、異なる実施形態について、必要であるように、あるいは所望するように、様々なステップを結合し、省略し、追加し、あるいは修正することができる。

【0126】ステップ330で、中央フレームワーク・インスタンスは、モジュール解放通知をユーザ・モジュールから受け取る。図16について先に述べたように、ユーザ・モジュールがサービス・モジュールを要求したとき、ユーザ・モジュールがサービス・モジュールのユーザのリストに追加される。ステップ332で、中央フレームワーク・インスタンスが、解放するユーザ・モジュールを、解放されるモジュールのユーザのリストから削除する。ステップ334で、フレームワーク・インスタンスは、他のユーザ・モジュールがこの解放されるモジュールを使用中であるかどうかを判定する。この判定は、たとえば、他のモジュールが解放モジュールのユーザ・モジュール・リスト内に存在するかどうかを検査することによって行う。そうである場合、実行が停止する。

【0127】他のモジュールが、解放されるモジュールを使用中でない場合、中央フレームワーク・インスタンスは、解放されるモジュールをアンロードしようと試みることができる。ステップ336で、フレームワーク・インスタンスは、解放されるモジュールのCanFinalizeメソッドを呼び出すことができる。CanFinalizeメソッドは、このモジュールをアンロードできる場合にtrueを返し、そうでない場合にfalseを返す。ステップ336で、CanFinalizeメソッドがfalseを返した場合、実行が停止する。そうでない場合、解放されるモジュールのFinalizeメソッドを呼び出すことができる。Finalizeメソッドは、リソースの解放など、モジュールのアンロードに必要ないかなるステップも実行することができる。次いで、このモジュールをアンロードすることができ、特定の実施形態に応じてこれにガベージ・コレクションなどを含めることができる。

【0128】本発明を特定の実施形態と共に説明したが、本明細書に述べた特定の形式に限定することを意図するものでなく、逆に、添付の特許請求の範囲によって定義された本発明の精神および範囲内に適度に含めるこ

とができるような代替物、修正および均等物を包含することを意図するものである。

#### 【図面の簡単な説明】

【図1】ブックマークを付けることができる様々なタイプのデータ・ソースを示すブロック図である。

【図2】特定のデータ・ソースを参照するブックマークを作成するプロセスを示すフローチャートである。

【図3】それぞれのデータ・ソースを参照するブックマークを使用するプロセスを示すフローチャートである。

【図4】JavaBeans™ 起動フレームワークアーキテクチャを含む主要な要素を示すブロック図である。

【図5】典型的な小フットプリント・デバイスのハードウェア・アーキテクチャを示すブロック図である。

【図6】コンテインメント・フレームワーク内のアプリケーションおよびサービスを実行するシステムに含まれているハードウェア/ソフトウェア層の典型的な階層を示す図である。

【図7】コンテインメント・フレームワーク内のアプリケーション/サービスを実行する小フットプリント・デバイスがローカルのサービス・ベースのネットワークに接続されている例示的ネットワークを示す図である。

【図8】サービス・プロバイダがルックアップ・サービスを見つける発見プロセスを示す図である。

【図9】サービス・プロバイダがそのサービスをルックアップ・サービスに登録する結合プロセスを示す図である。

【図10】クライアントがルックアップ・サービスからサービスを要求するルックアップ・プロセスを示す図である。

【図11】クライアントがルックアップ・サービスから受け取ったサービス・オブジェクトを使用してサービス呼び出しプロセスを示す図である。

【図12】コンテインメント・フレームワークの基本アーキテクチャを示す抽象的なブロック図である。

【図13】コンテインメント・フレームワーク内のモジュール要求リスナを使用して階層コンテインメント環境をシミュレートすることを示す図である。

【図14】コンテインメント・フレームワーク内のモジュール要求リスナを使用して階層コンテインメント環境をシミュレートすることを示す図である。

【図15】パーセルを使用してモジュールを共にグループ化することを示す図である。

【図16】中央フレームワーク・インスタンスがクライアント・モジュールからサービス・モジュールのルックアップ要求を受け取ったときに実行することができる典型的なルックアップ・プロセスを示すフローチャートである。

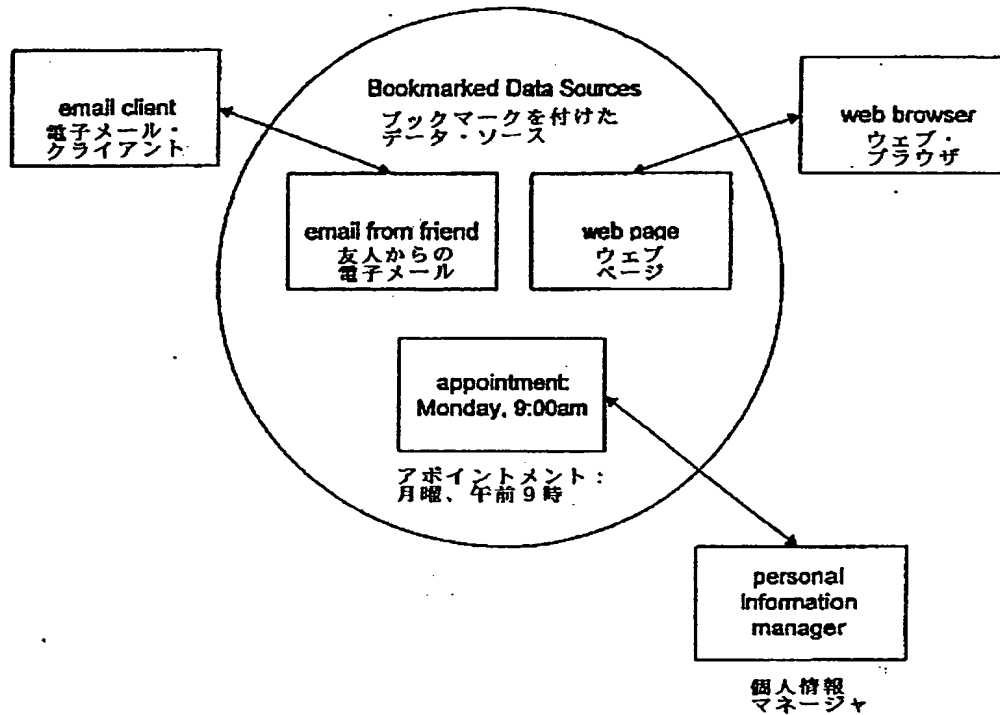
【図17】モジュール解放プロセスを示すフローチャートである。

【符号の説明】

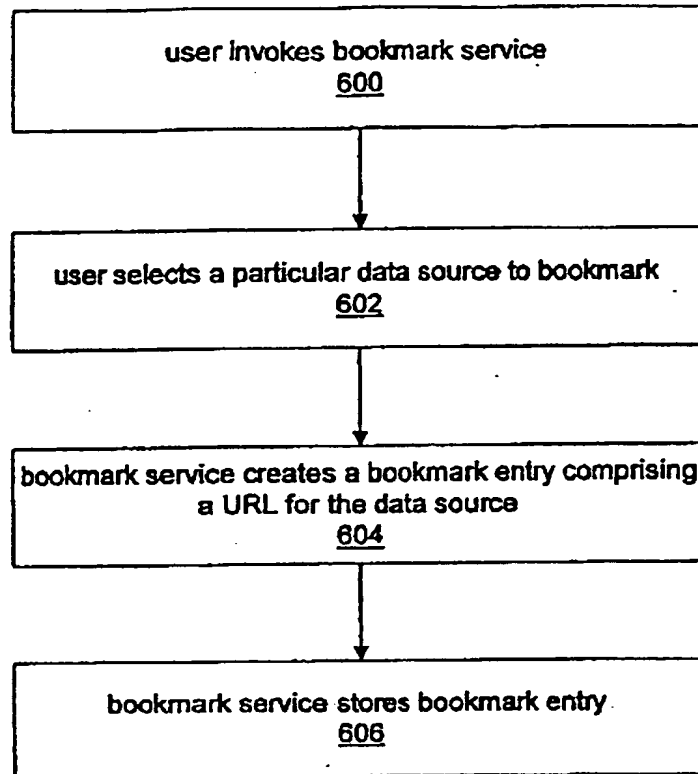
130 プリンタ  
 132 インターネット・テレビ  
 134 「高性能」携帯電話  
 136 ルックアップ・サービス  
 138 プリント・サービス  
 140 Webサービス  
 142 コンテインメント・フレームワーク・インタフ

エース  
 144 コンテインメント・フレームワーク  
 146 ルックアップ・サービス  
 148 Webブラウザ  
 150 電子メール・クライアント  
 152 パーソナル・オーガナイザ  
 154 インターネット

【図1】



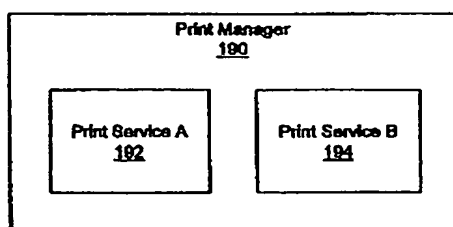
【図 2】



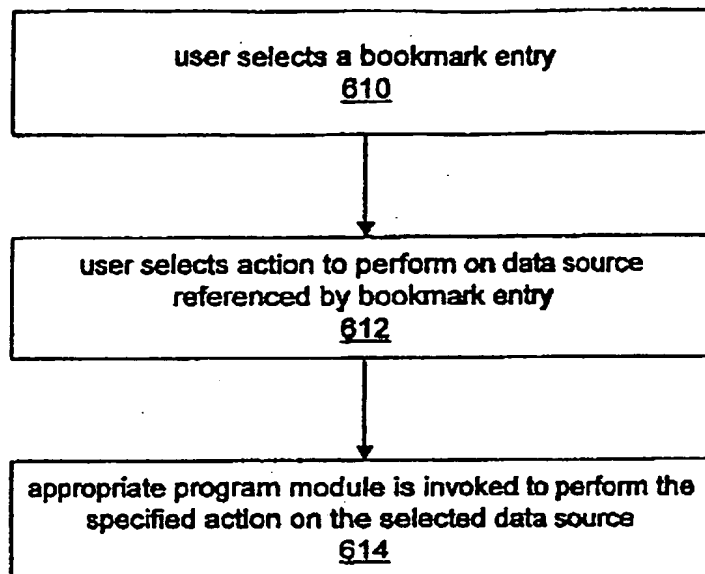
- 600 ユーザがブックマーク・サービスを呼び出す  
602 ユーザがブックマークを付ける特定のデータソースを選択する  
604 ブックマーク・サービスが、データ・ソースのURLを含む  
ブックマーク・エントリを作成する  
606 ブックマーク・サービスがブックマーク・エントリを格納する

【図 13】

- 190 プリント・マネージャ  
192 プリント・サービスA  
194 プリント・サービスB

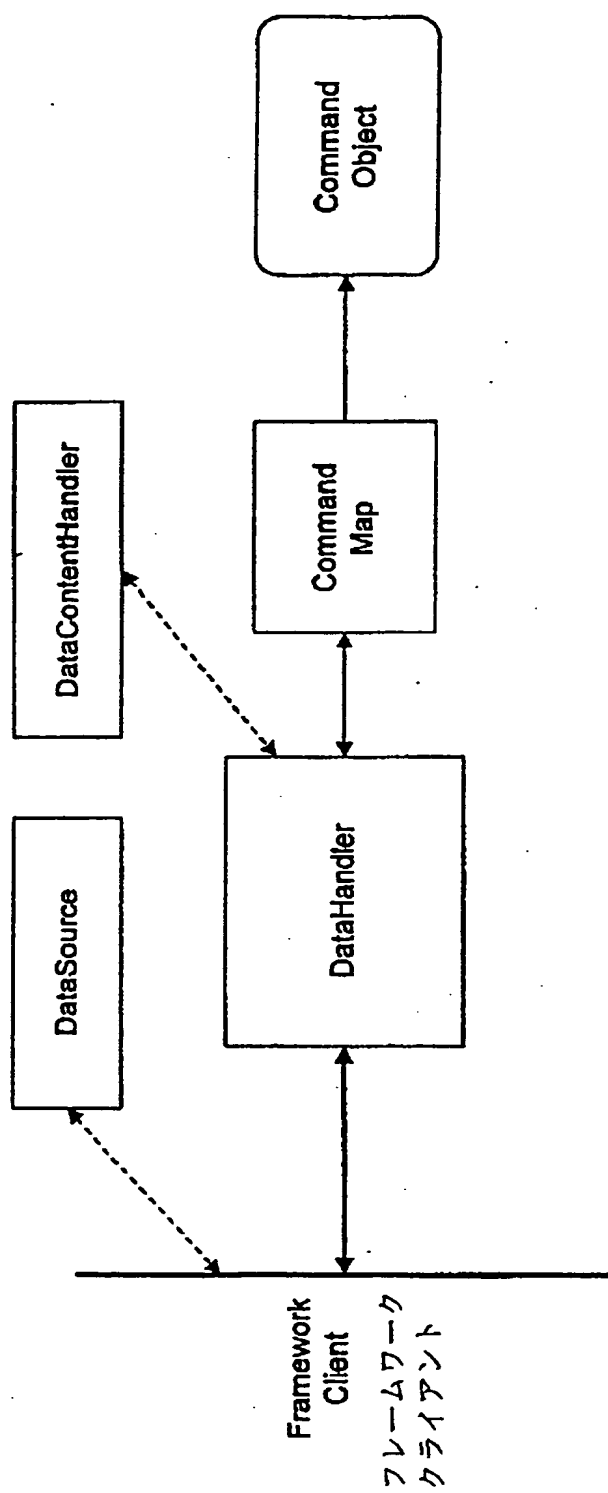


【図 3】

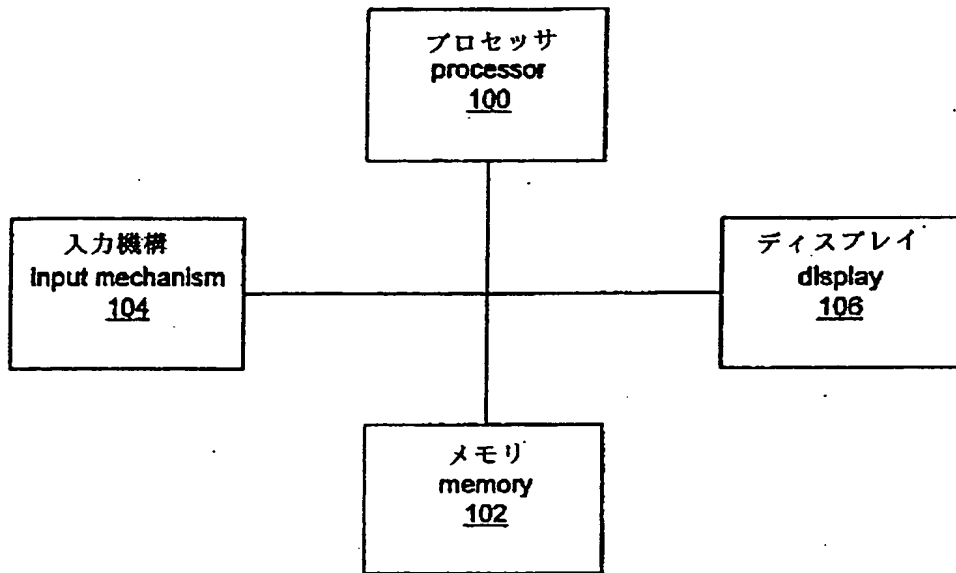


- 610 ユーザがブックマーク・エントリを選択する
- 612 ユーザが、ブックマーク・エントリによって参照されたデータ・ソースに対して実行するアクションを選択する
- 614 指定したアクションを、選択したデータ・ソースに対して実行するために、適切なプログラム・モジュールを呼び出す

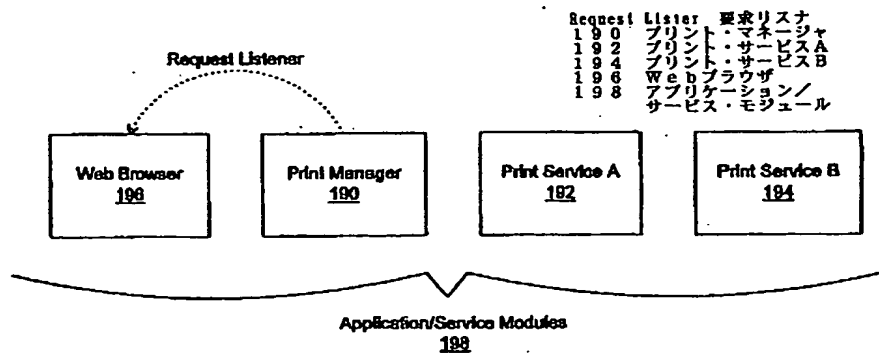
【図 4】



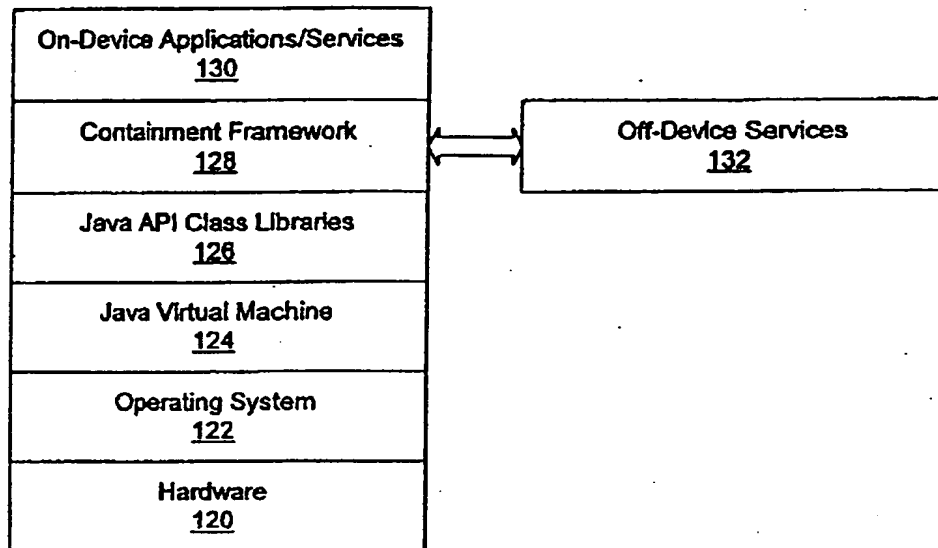
【図 5】



【図 14】



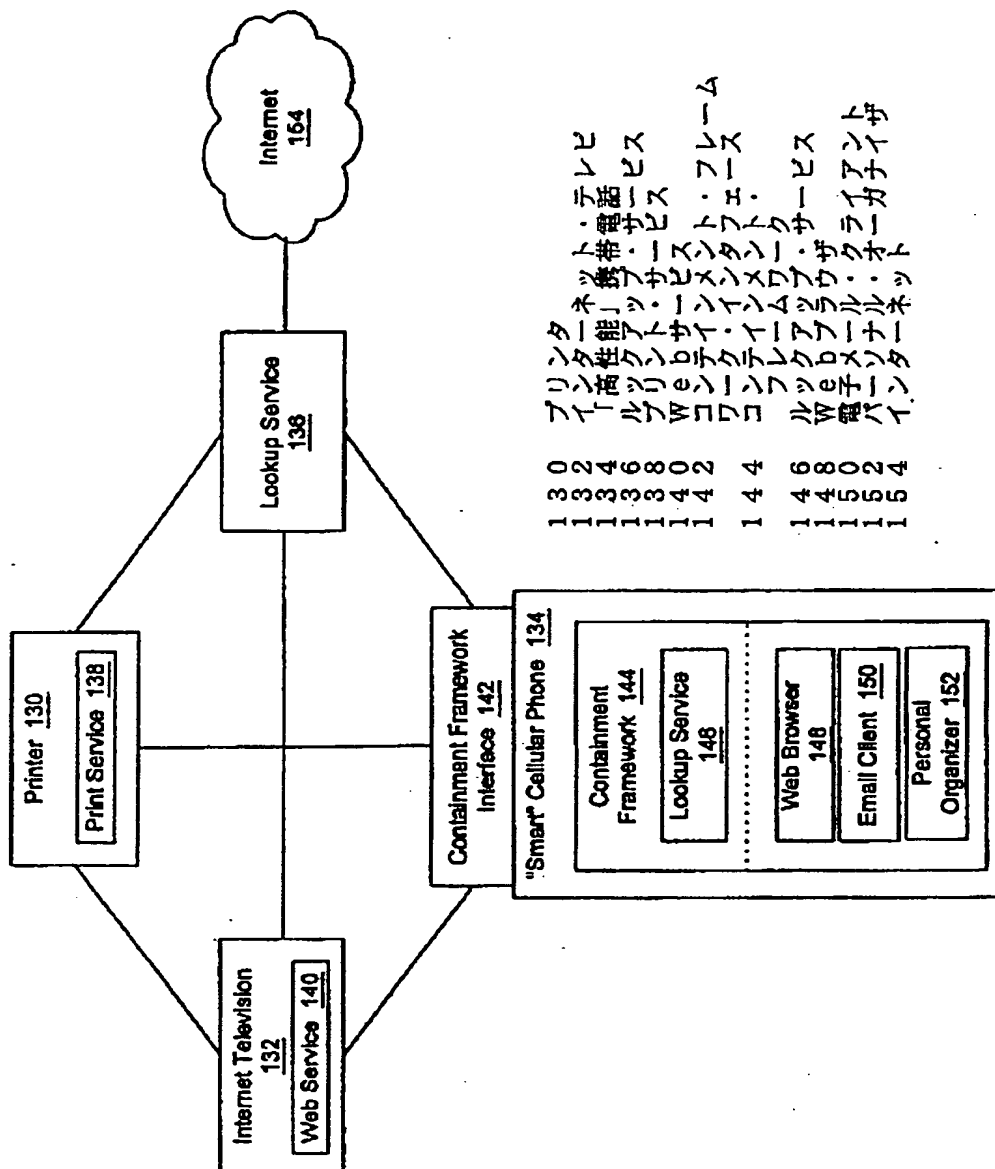
【図 6】



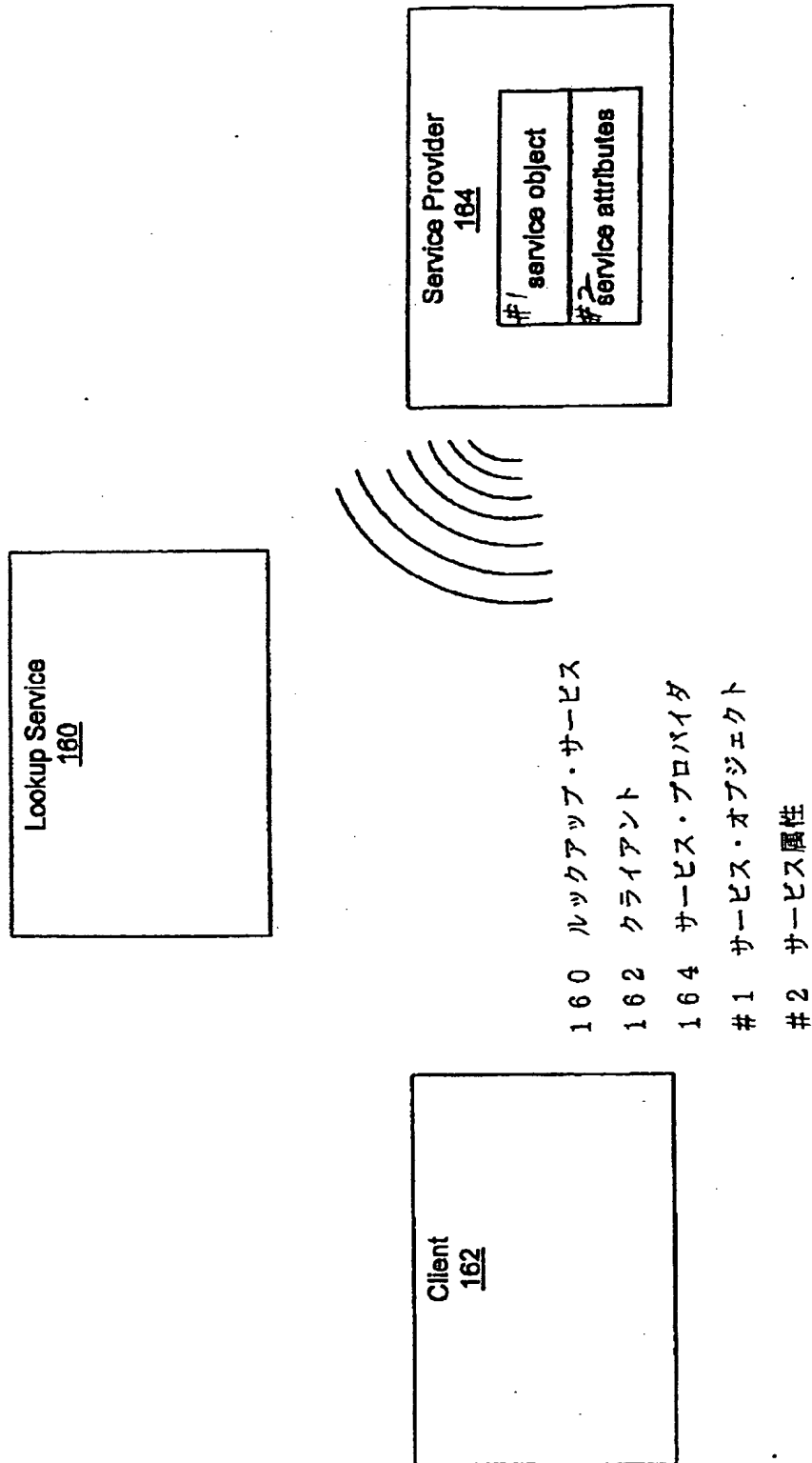
- 120    ハードウェア
- 122    オペレーティング・システム
- 124    Java仮想マシン
- 126    Java APIクラス・ライブラリ
- 128    コンテインメント・フレームワーク
- 130    オン・デバイス・アプリケーション／サービス
- 132    オフ・デバイス・サービス



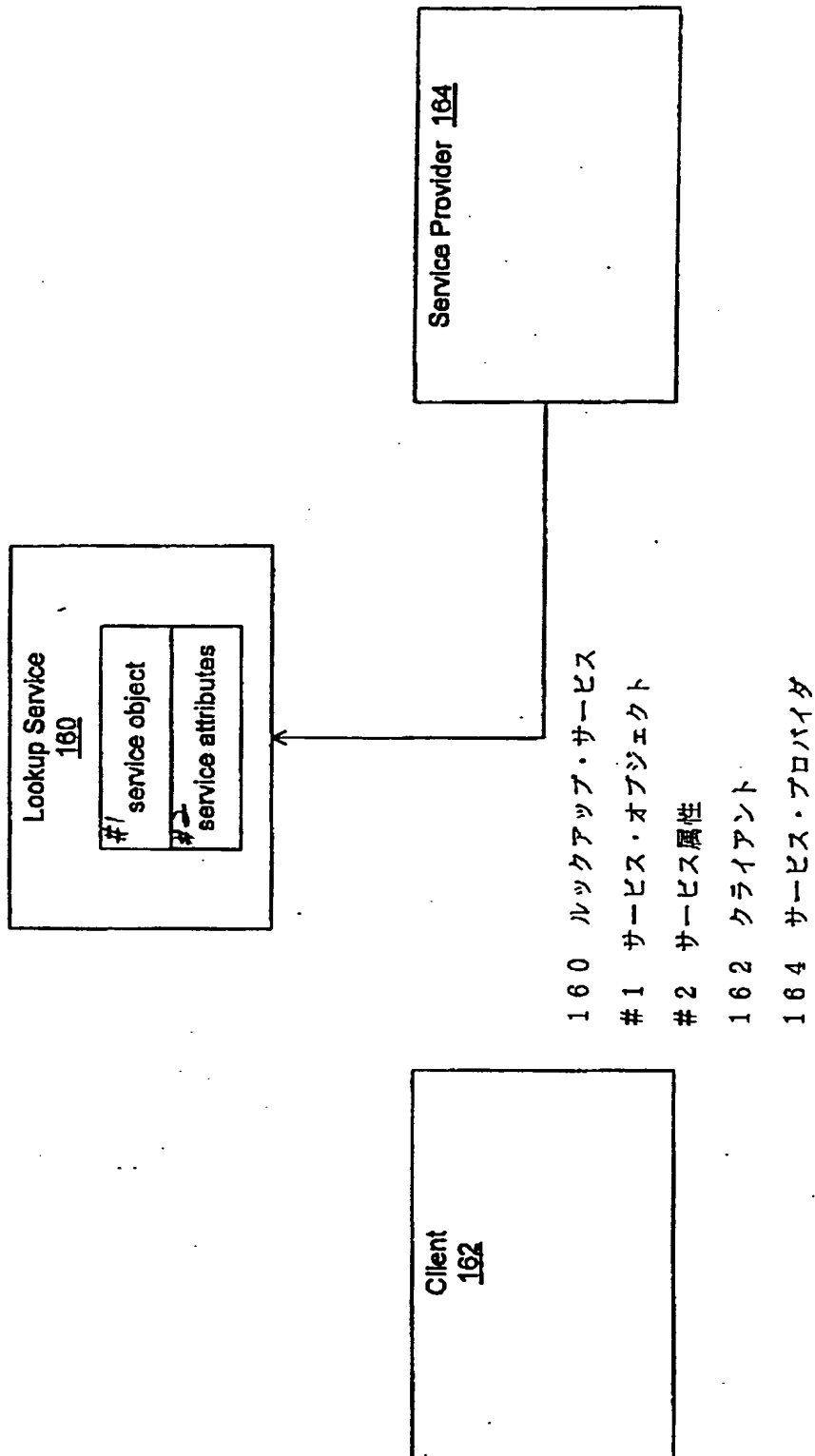
【图 7】



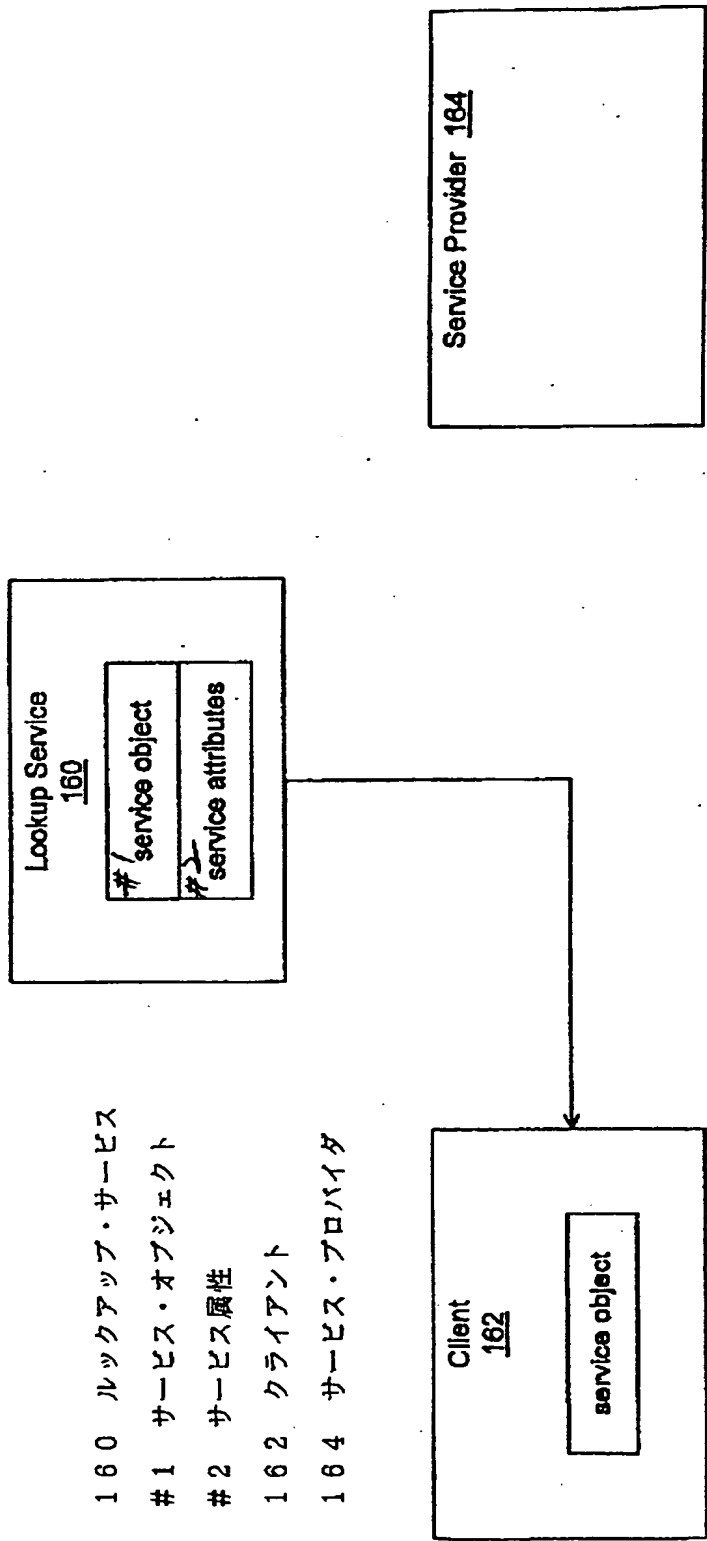
【図 8】



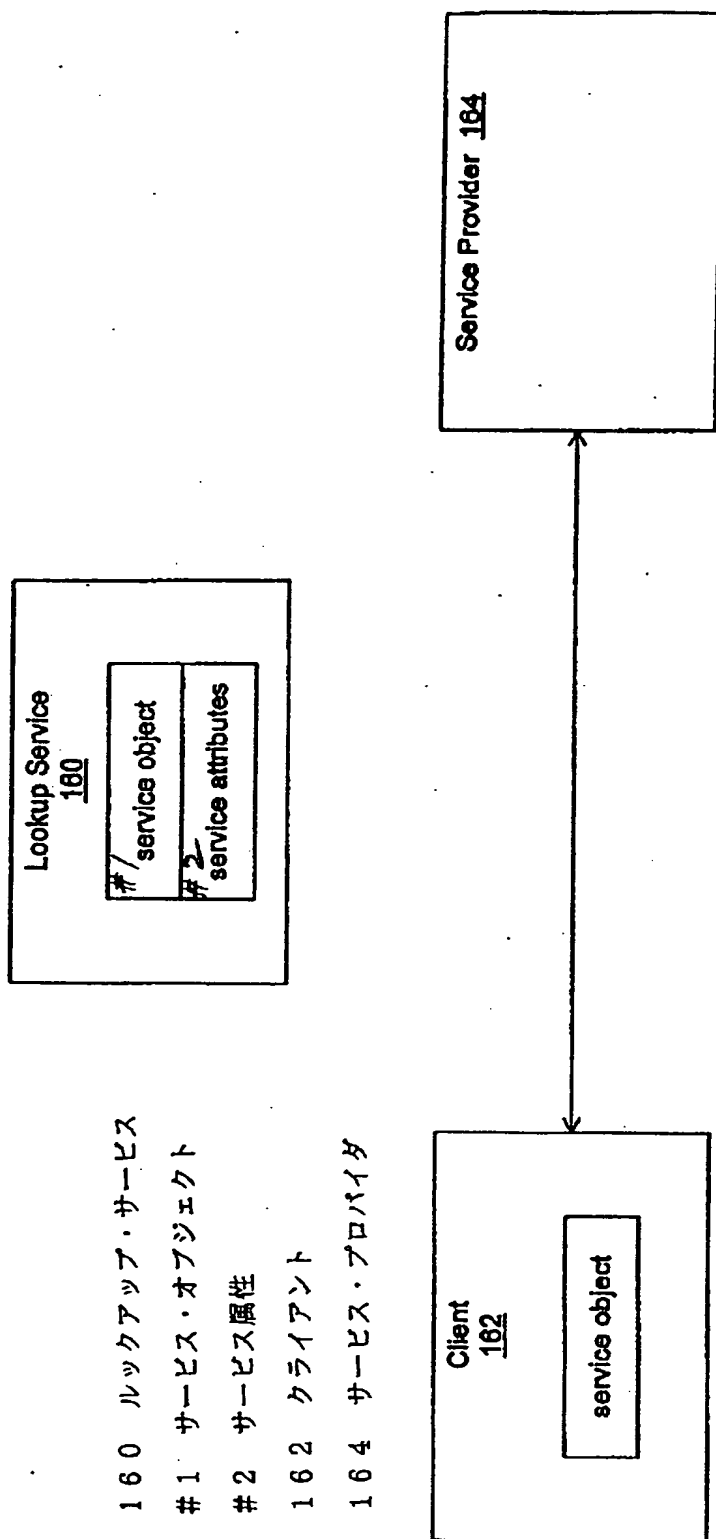
【図 9】



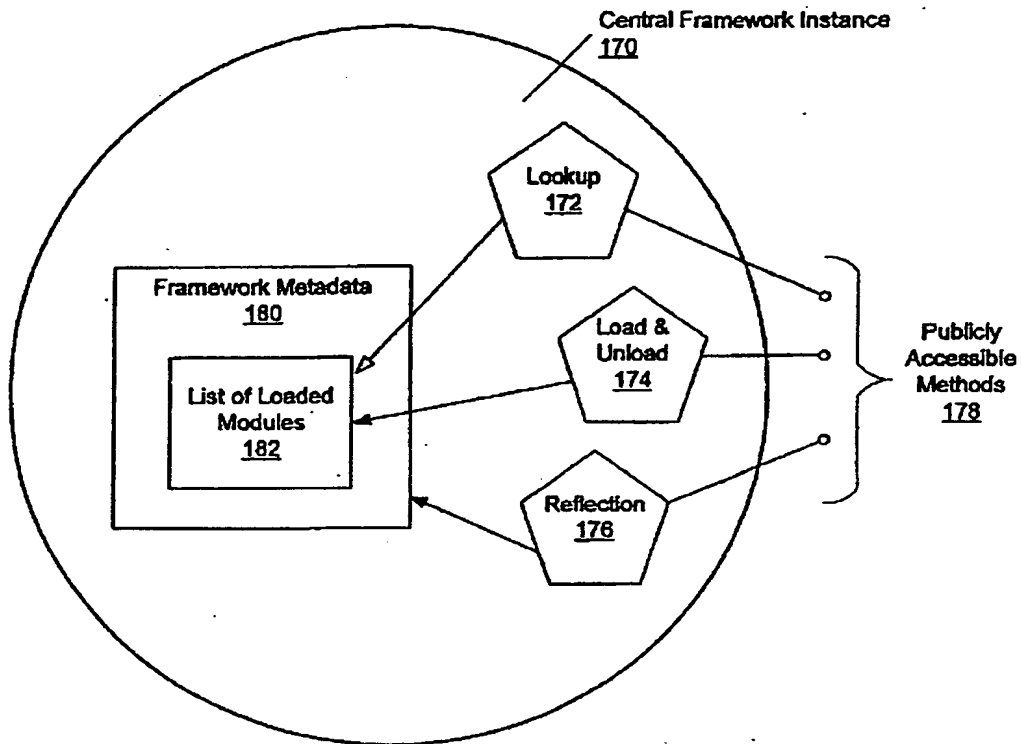
【図10】



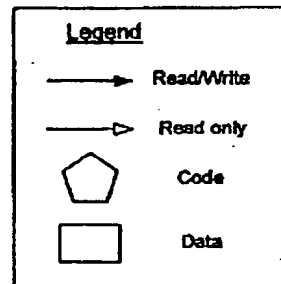
【図 11】



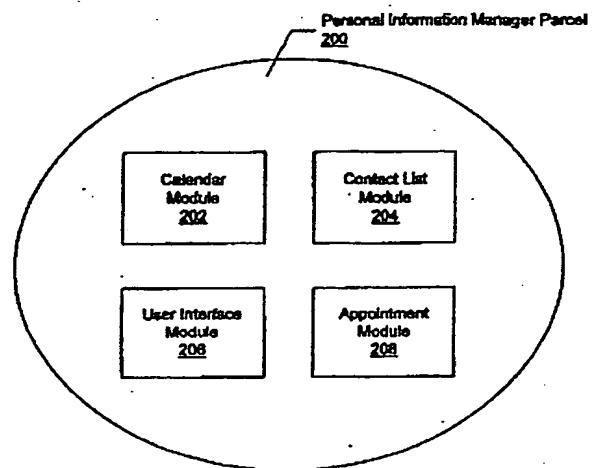
【図 12】



170 中央フレームワーク・  
 172 インスタンス  
 174 ロードおよびアンロード  
 176 リフレクション可能なメソッド  
 178 公開されたアクセス可能なメソッド  
 180 フレームワーク・メタデータ  
 182 ロードされたモジュールのリスト  
 Legend  
 Read/Write 読み書き専用  
 Read only 読み専用  
 Code  
 Data

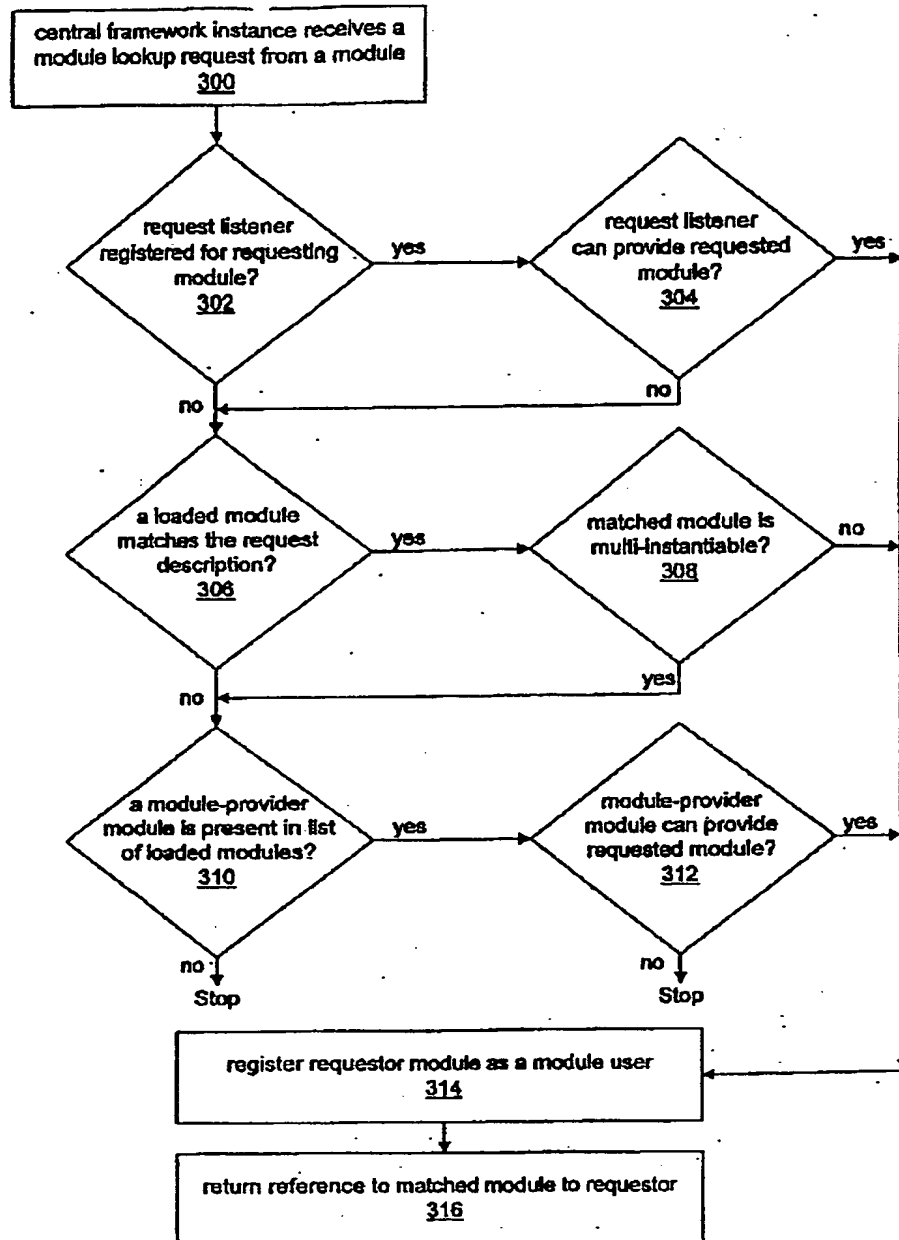


【図 15】



- 200 個人情報マネージャ・パーセル
- 202 カレンダ・モジュール
- 204 コンタクト・リスト・モジュール
- 206 ユーザ・インタフェース・モジュール
- 208 アポイントメント・モジュール

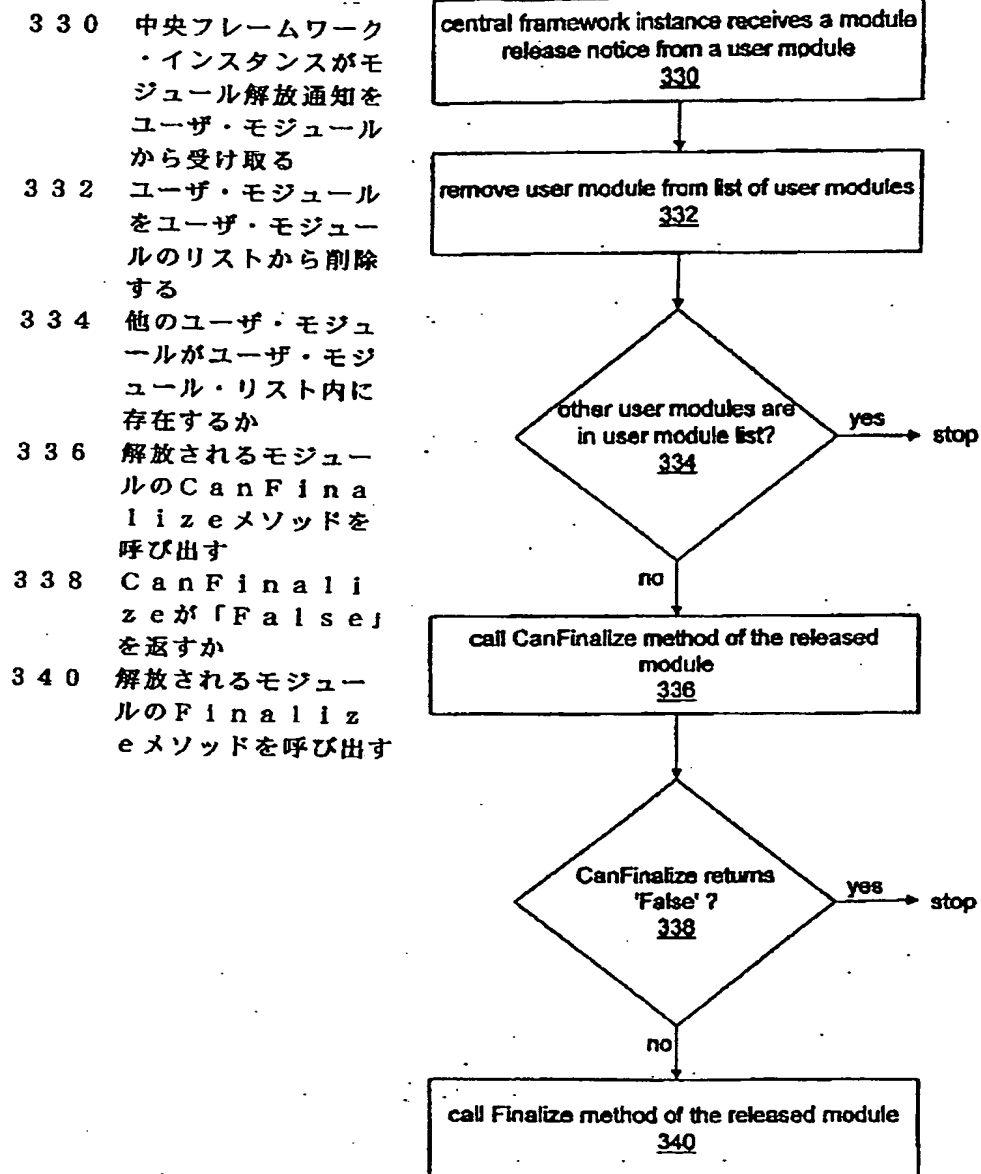
【図 16】



300 中央フレームワークインスタンスがモジュール・ユーザとして登録する  
 302 フレームワークが要求されたモジュールをモジュール・ユーザとして登録する  
 304 リクエストされたモジュールが要求されたモジュール・ユーザとして登録する  
 306 一致したモジュールが要求されたモジュール・ユーザとして登録する  
 308 そのリスト内からモジュールが要求されたモジュール・ユーザとして登録する  
 310 モジュール・ユーザとして登録する  
 312 モジュール・ユーザとして登録する  
 314 一致したモジュールが要求されたモジュール・ユーザとして登録する  
 316 一致したモジュールが要求されたモジュール・ユーザとして登録する



【図 17】



## 【手続補正書】

【提出日】平成12年10月13日（2000. 10. 13）

## 【手続補正1】

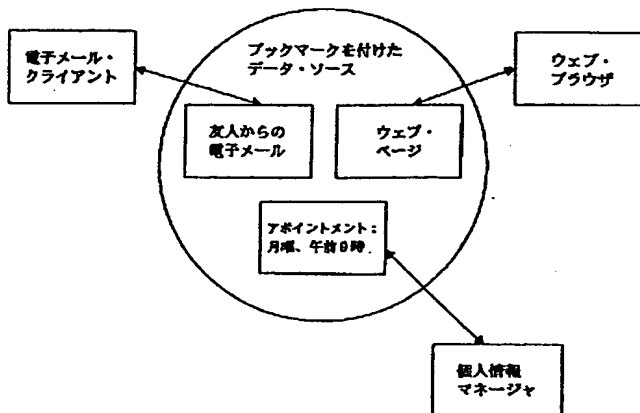
【補正対象書類名】図面

【補正対象項目名】全図

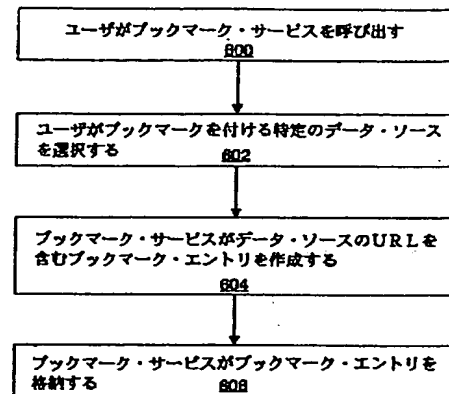
【補正方法】変更

【補正内容】

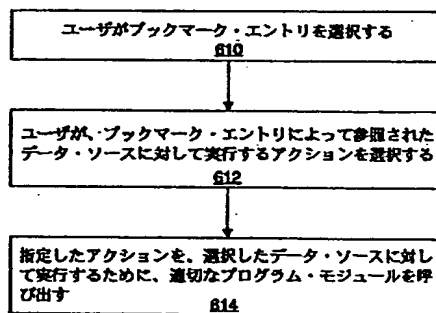
【図 1】



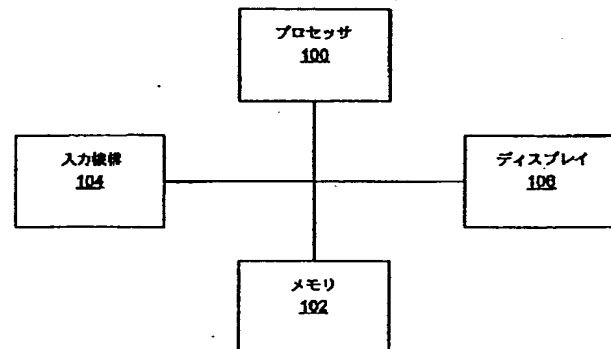
【図 2】



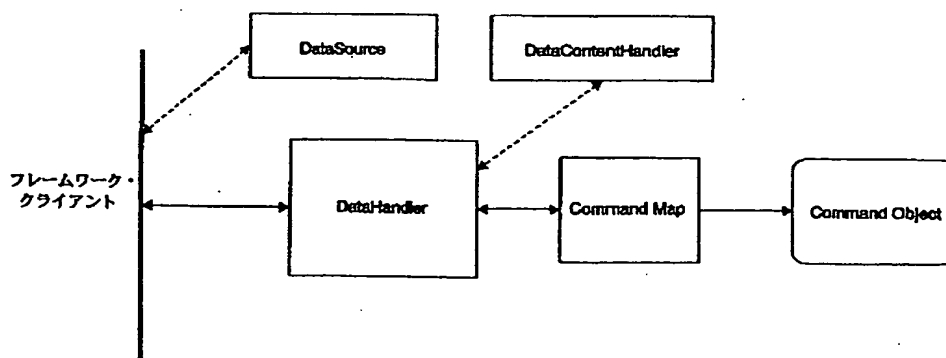
【図 3】



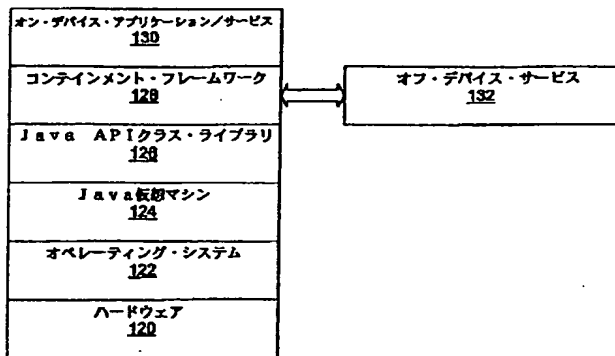
【図 5】



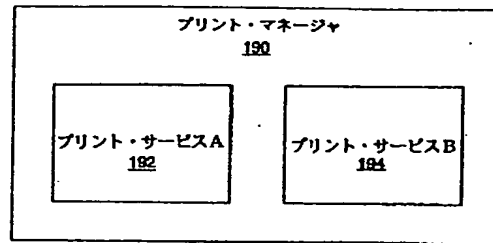
【図 4】



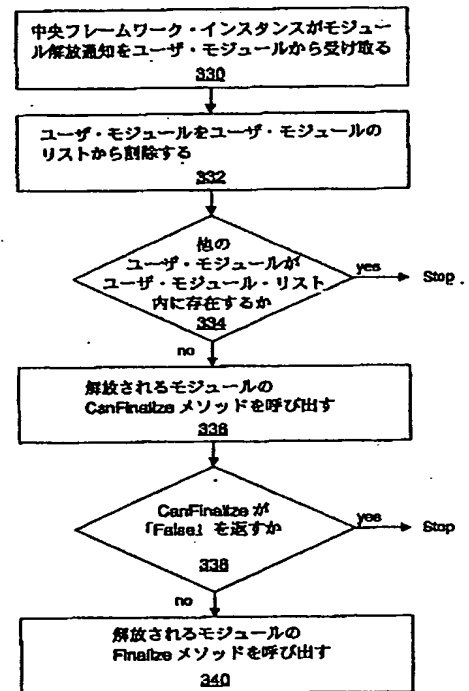
【図 6】



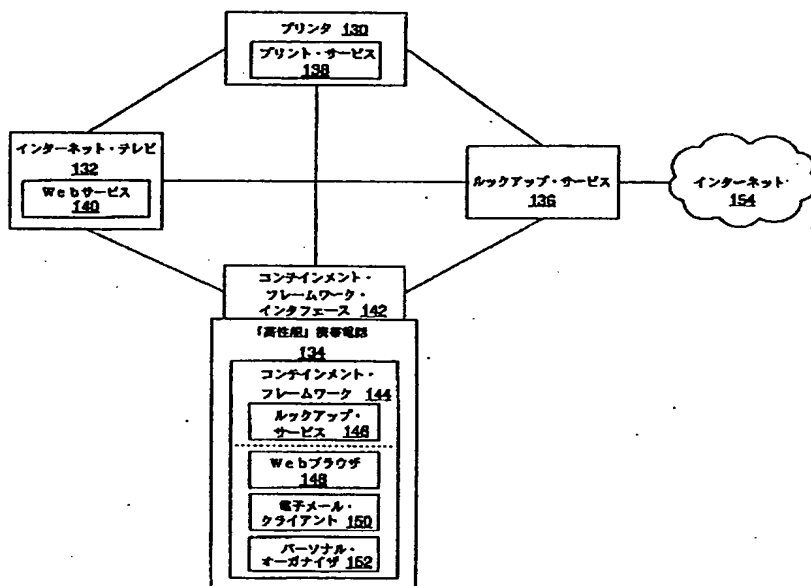
【図 13】



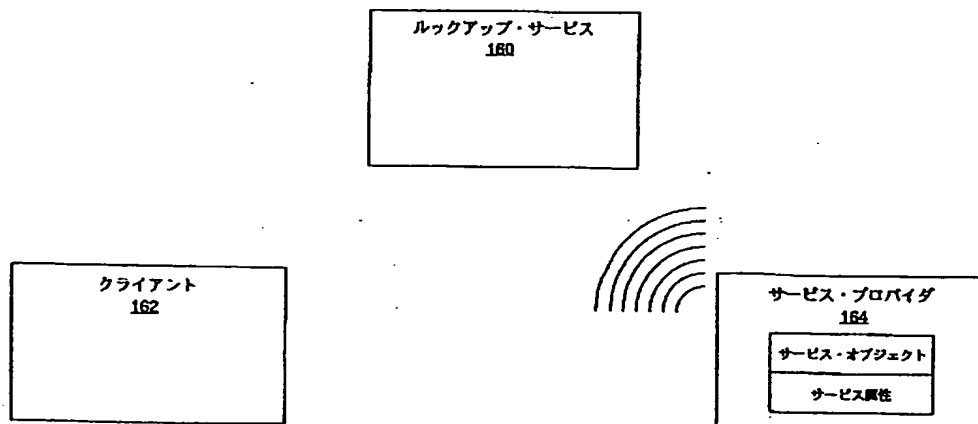
【図 17】



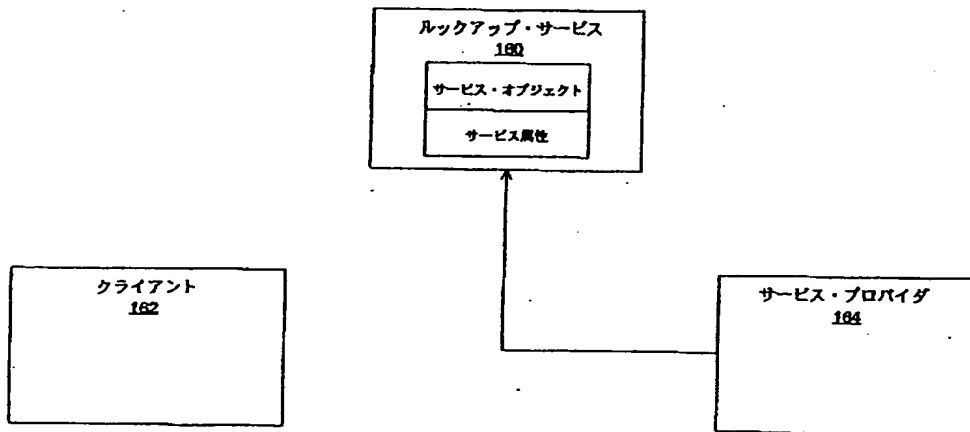
【図 7】



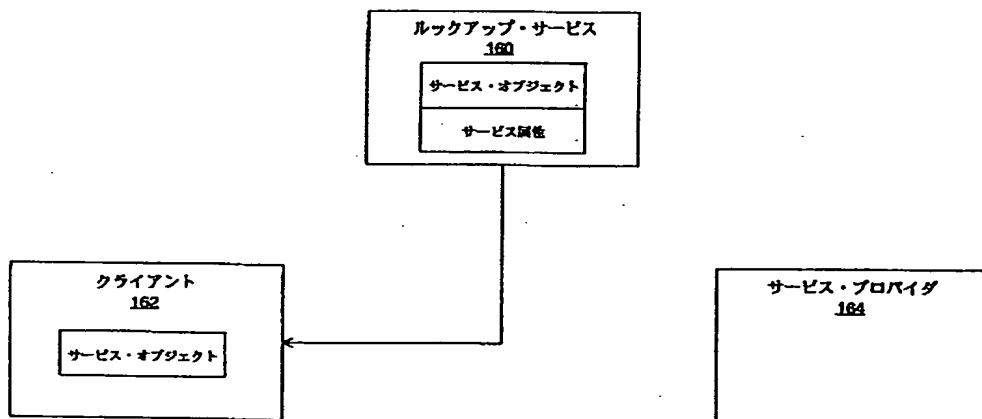
【図 8】



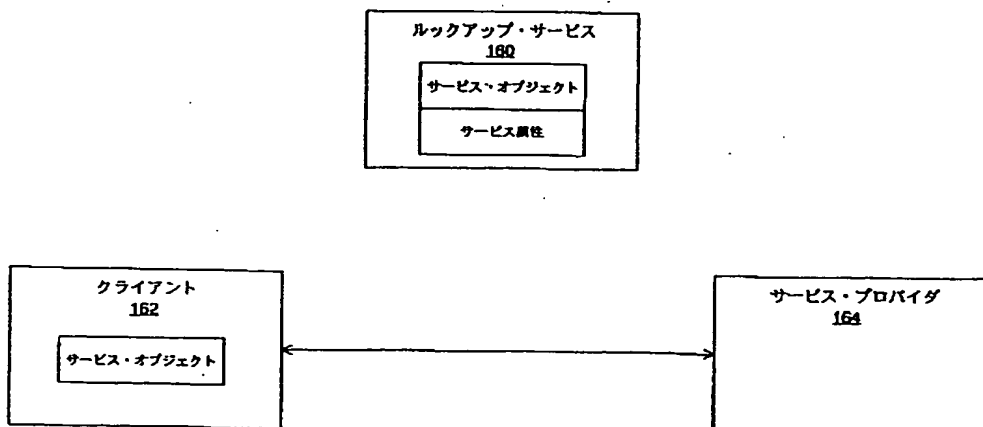
【図 9】



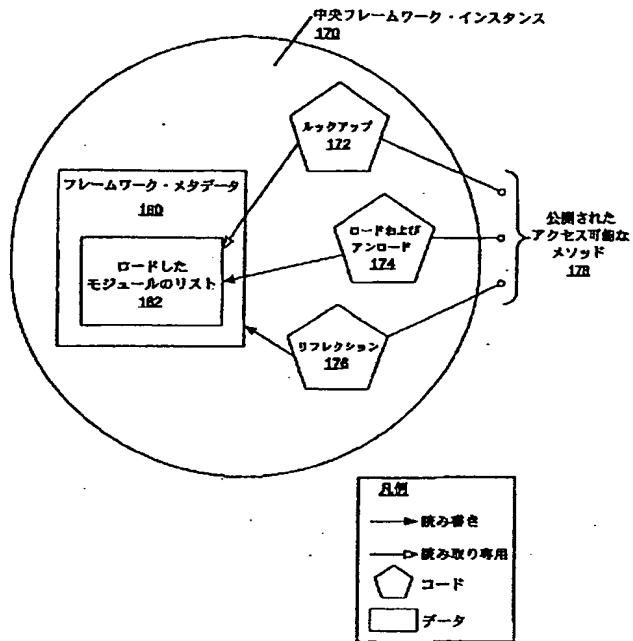
【図 10】



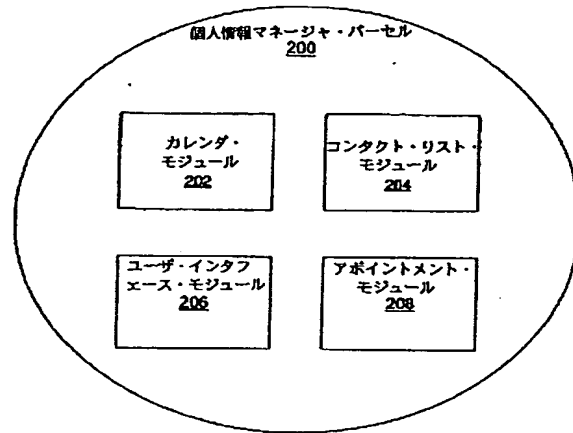
【図 11】



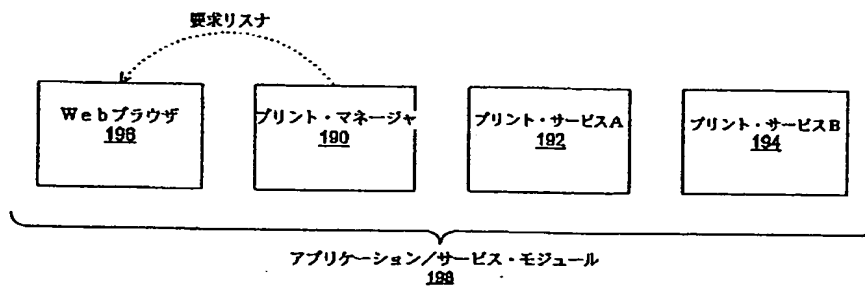
【図 12】



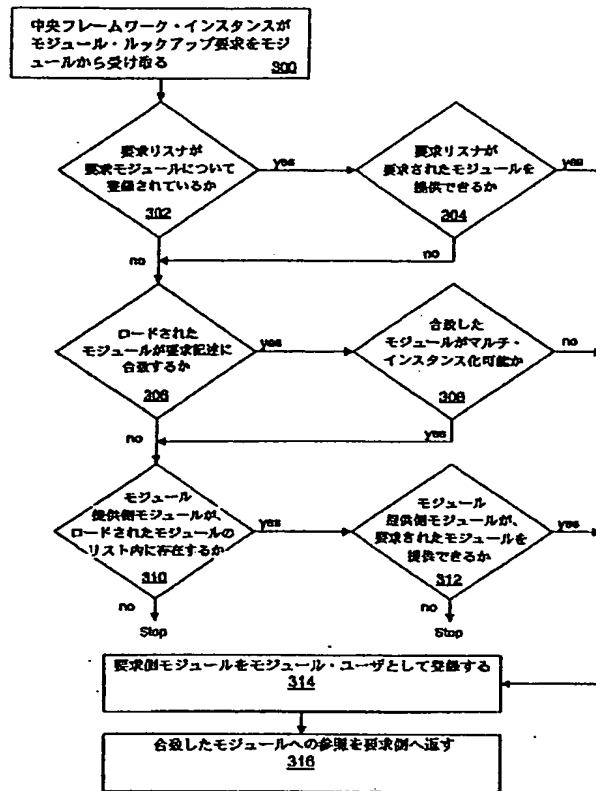
【図 15】



【図 14】



【図 16】



フロントページの続き

(72)発明者 マーク・ベイカー

カナダ国・ケイ 1 エム 2 シイ 5 ・オンタ  
リオ州・オタワ・マッケイ ストリート・

399